Universitat de Lleida
Escola Politècnica Superior
Enginyeria en Informàtica

Sistemes Informàtics (Treball de Final de Carrera)

Sistema de monitorización para la plataforma CompP2P

Autor: **Hector Blanco de Frutos**

Director: **Fernando Cores Prado**

Enero de 2008

Índice

Prólogo

En los últimos años se ha visto un cambio muy importante en el campos de la computación de altas presentaciones, en el que los grandes supercomputadores eran la única herramienta a emplear. Esto ha cambiado con la adopción de nuevas tecnologías y filosofías de computación, que intentan aprovechar los recursos ociosos de miles o incluso millones de ordenadores de sobremesa estándares, para realizar las tareas que antes realizaban en exclusiva los mencionados supercomputadores.
Una de las filosofías que vienen a cambiar la concepción de computación distribuida de altas prestaciones es la arquitectura Peer-to-Peer, que permite definir redes de cómputo con un número indefinido de ordenadores de sobremesa interconectados entre si sin la necesidad de una infraestructura dedicada a tal propósito.

En este ámbito de la computación Peer-to-Peer se emplaza éste proyecto. Sus objetivos son los de incorporar mecanismos de monitorización a la plataforma CompP2P desarrollada en la Universitat de Lleida [GR06], así como el de desarrollar un conjunto de herramientas y aplicaciones que puedan implementar y emplear esos mecanismos de monitorización para ser empleados en un entorno de producción.

La memoria del proyecto se divide en los siguientes puntos principales.

El el primer capítulo se hará una Introducción a los entornos de computación de altas presentaciones, herramientas de monitorización, y a los entornos de computación distribuida Peer-to-Peer.

En el segundo capítulo se explicará en detalle el diseño de la arquitectura de la plataforma CompP2P, así como su funcionamiento básico, su gestión de nodos y recursos, y la realización de tareas para ser ejecutadas en la propia plataforma.

En el tercer capítulo se detallarán los objetivos principales y los requisitos necesarios para poder cumplir esos objetivos. Tambien se explicarán las principales decisiones y aspectos relativos al diseño de los mecanismos de monitorización y a las herramientas que posteriormente se implementarán.

En el cuarto capítulo se explicarán los aspectos principales relativos a la implementación de los mecanismos de monitorización y a las herramientas que emplearan esos mecanismos. También se explicarán partes concretas del código implementado que sean de gran relevancia.

En el quinto capítulo se mostrarán los resultados y gráficas de las pruebas realizadas para probar el funcionamiento de las diferentes herramientas y mecanismos implementados, así como el de realizar comparativas y análisis de estos resultados.

En el sexto capítulo se detallarán las conclusiones del proyecto, así como analizar y exponer las posibles lineas abiertas y posibles futuras mejoras en las tareas realizadas.

Por último se incluye un conjunto de apéndices se incluirán el índice de ilustraciones, bibliografía, referencias a otros documentos y trabajos, y la inclusión completa de los códigos fuente que se implementen en el proyecto.

1. Introducción

La idea principal de la computación distribuida es la de coger problemas que son intratables para un único ordenador y dividirlos en problemas más pequeños que puedan ser tratados por múltiples ordenadores.

En los últimos años se ha visto un avance importante en la arquitectura *peer to peer*, o *p2p*, que se basa en la idea de descentralizar los diferentes elementos del sistema. Un buen ejemplo de estas arquitecturas es la popular aplicación *emule*, la cual permite compartir archivos entre nodos que se conecten a una red de una forma prácticamente descentralizada, en la que los servidores solo actúan como enlaces entre los nodos.

A simple vista se puede ver las ventajas que este tipo de arquitecturas pueden aportar a la computación distribuida. Una de ellas, y tal vez la principal es el no necesitar de un servidor central, como sí ocurre en las arquitecturas cliente-servidor, y en las que si cae este servidor, se ve paralizado el trabajo de toda la red.
Las arquitecturas peer-to-peer se pueden distribuir en dos grandes grupos: puras e híbridas. Un diseño peer-to-peer puro implica que todos los nodos del sistema son autosuficientes para gestionar los recursos que se proporcionan a la red. En cambio, una arquitectura peer-to-peer híbrida toma algunas partes de las arquitecturas cliente-servidor para su diseño. Generalmente estas arquitecturas híbridas están estructuradas de forma que una serie de servidores o nodos centrales se encargan de gestionar o indexar los recursos de la red que aprovecharán los nodos simples.
Cada uno de los diseños tiene ventajas e inconvenientes, y dependiendo de su ámbito de utilización serán más o menos convenientes. Por ejemplo, las arquitecturas híbridas proporcionan un sistema de gestión de recursos más sencillo que las arquitecturas puras, ya que en las primeras son los nodos centrales o servidores los que se encargan de esto, mientras que en éstas últimas los mecanismos para gestionar los recursos son mucho más complejos, ya que debe realizarse entre todos los nodos. Por el contrario, las arquitecturas puras generalmente proporcionan un grado de anonimato y seguridad superiores al no haber grandes servidores o nodos centrales en los que se indexe y almacene la información de los recursos que se comparten en la red. Ejemplos de

arquitecturas híbridas serian las redes edonkey2000 y Napster, y como arquitecturas puras tendríamos a las redes FreeNet y Kademlia.

En las arquitectura *peer to peer* el hecho de perder un nodo no tiene que implicar la caída de todo el sistema. Dependiendo de la forma en que esté diseñado el sistema puede implicar simplemente el perder una pequeña parte del trabajo que se estuviera realizando en ese momento. Otra gran ventaja está en el echo que todos los nodos del sistema pueden realizar las mismas tareas, al no haber nodos más importantes que otros, lo que lo convierte en un sistema distribuido auto-organizado.
Finalmente, en esta última categoría de los sistemas distribuidos basados en la arquitectura *peer to peer* es a la que pertenece compP2P, el entorno de cómputo distribuido que será tratado a lo largo de esta memoria.

2. Plataforma CompP2P

En éste capítulo se introducirán los aspectos y características principales de la plataforma CompP2P, así como su arquitectura interna.

2.1 Introducción a CompP2P

La plataforma compP2P es un entorno de cómputo distribuido desarrollado en Java que se basa en la arquitectura peer-to-peer (p2p) para lanzar trabajos que serán ejecutados remotamente por diferentes nodos.
Este entorno distribuido fue desarrollado como proyecto de final de carrera en la EPS [GR06].
La plataforma se comporta como un entorno peer-to-peer híbrido, en el que los nodos comparten recursos de cómputo. Es un entorno híbrido porqué no todos los nodos actuan igual, ya que una parte de ellos tendrán un rol más importante, el de gestionar la conexión y recursos de una parte de los nodos del sistema, dividiendo la red en grupos de varios nodos. En redes peer-to-peer puras no existe esa división, y todos los nodos tienen el mismo rol en todo el entorno [PEER02].

2.2 Entorno CompP2P

La arquitectura sobre la que está construido el entorno CompP2P está diseñada en varios niveles, siendo el más bajo Java, y por encima de este, JXTA, que es el framework que pone en funcionamiento todos los sistemas de distribución de nodos en la red, y que establece los mecanismos para comunicarlos entre sí.

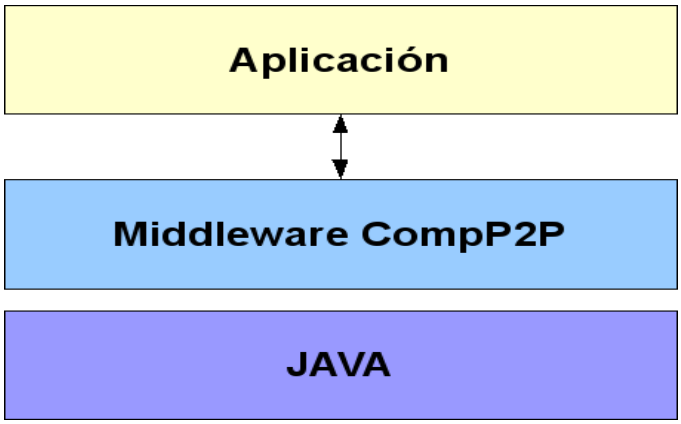


Ilustración 1: Diagrama básico del entorno CompP2P

2.3. Java Environment

CompP2P ha sido programado enteramente en Java por un motivo capital, y es el de su capacidad de potabilidad. Había que escoger un lenguaje de programación que pudiera ser usado también en diferentes plataformas de hardware y software. Java cumple con este requisito a la perfección, ya que la ejecución de los programas no se ejecutan directamente en código máquina sobre el hardware usuario, sino que estos programas son generados en forma de bytecode independiente de plataforma, que luego interpreta la máquina virtual de Java en la plataforma final del usuario.
Otra gran cualidad de Java es su versatilidad y potencia a la hora de trabajar con objetos, cualidad que es de gran utilidad para la plataforma CompP2P, ya que permite que se pueda transmitir entre los nodos objetos de datos directamente serializados, sin necesidad de realizar conversiones ni adaptaciones específicas para cada nodo.

2.4 JXTA

JXTA es una plataforma p2p de código abierto desarrollada mayoritariamente por Sun Microsystems, y que ha sido portada a diferentes lenguajes de programación, como Java y C/C++.
En la plataforma se definen una serie de protocolos basados en XML que permiten a distintos dispositivos de una red intercambiar mensajes entre si, independientemente de la topología de la red y de la plataforma del dispositivo, como Pcs, *handhelds* ...etc. Es la utilización de XML como estructura de los mensajes lo que le permite esta facilidad de portabilidad a lenguajes de programación y la interacción entre tan variados dispositivos y plataformas.
Los peers en *JXTA* se definen un entorno de red virtual, lo que les permite comunicarse entre si incluso si se encuentran detrás de un cortafuegos (firewall). Ésto, sumado al incorporar identificación de cada recurso mediante el uso del sistema de hash de 160 bits ,SHA-1 URN, hace posible que un nodo pueda mantener su identificación pese a cambiar su ubicación en la red.

A continuación se expondrán detalladamente los componentes principales de la arquitectura de JXTA.

2.4.1 Endpoints

Un endpoint es una entidad emisora o receptora del cualquier conjunto de datos són retransmitidos en una red de peers. Un endpoint proporciona abstracción de las interfícies de red usadas para enviar y recibir datos, sin necesidad de tener en cuenta a aspectos como protocolos o la arquitectura del sistema.

2.4.2 Anuncios

Un anuncio o *advertisement* en JXTA es un documento XML que describe cualquiera de los recursos de una red P2P (grupos, pipes, peers, servicios ...etc). La comunicación en JXTA se puede realizar mediante el intercambio de uno o varios de estos documentos XML.
Los anuncios se utilizan para presentar en la red todos los recursos que en ella hay disponibles, y la ubicación de cada uno de los mismos. Es una forma de simplificar la organización de las redes de peers.

2.4.3 Pipes

Los pipes son canales virtuales de comunicación unidireccional y asíncronos que conectan a dos o más *endpoints.* Se usan para el intercambio de mensajes y datos, y se dividen en los siguientes tres tipos [JXP] (1.4.1):

- **Unicast** (JxtaUnicast) : Utilizado para enviar mensajes directamente de un peer a otro. No proporciona ni seguridad ni confiabilidad.
- **Unicast seguro** (JxtaUnicastSecure: Es una extensión de JxtaUnicast, con la diferencia que se usa conexión virtual segura mediante TLS entre los endpoints.
- **Propagate** (JxtaPropagate): Se usa para enviar mensaje de un peer a varios. Cualquier peer que tenga creado un pipe de entrada declarado como propagate, recibirá el mensaje enviado a ese pipe.

2.4.4. Peers

Los peers son los nodos dentro de la red P2P. En el caso de CompP2P, estos nodos se corresponden con los elementos distribuidos de cálculo. Normalmente un peer en una red p2p viene asociado a una única aplicación funcionando en una máquina.
Dentro de JXTA y CompP2P tenemos varios tipos de peer según cual sea su utilidad:

- **Simple peers:** Sirve a un solo usuario, y normalmente se encuentran detrás de cortafuegos, separado de la red. Debido a esto, posiblemente los peers fuera de un

cortafuegos no podrán comunicarse con *simple peers* que estén detrás de un cortafuegos, y por lo tanto son los peers con responsabilidad más baja en las redes JXTA.

- **Rendez-vous peers:** Un peer de este tipo se encarga de proporcionar a los otros peers una localización en la red y para descubrir otros peers y los recursos de estos. Los *rendez-vous peers* almacenan información de los peers y la ponen a disposición del resto, además de encargarse de propagar los mensajes entre las redes.
- **Router peers:** Se encarga de proveer a los peers de un mecanismo para comunicarse con otros peers que estén en redes detrás de un cortafuegos o de una infraestructura que utilice NAT. Para enviar mensajes a través de *router peers*, un peer deberá especificar con el mensaje, qué *router peer* deberá ser utilizado. Es un mecanismo que viene a substituir DNS.

2.4.5. Peer Groups

Es el nombre dado al mecanismo por el cual los peers pueden ser agrupados en grupos independientes, para servir todos a un interés común.
JXTA realiza esta separación por los siguientes motivos:

- La aplicación en la que han de colaborar unos peers necesita que estos sean separados del resto. Por ejemplo, por motivos de privacidad con los datos.
- Por requisitos de seguridad de sus peers, usando mecanismos de autenticación que restrinjan el acceso a otros peers.
- La necesidad de la información de estado sobre los miembros del grupo. Es decir, que unos peers puedan supervisar a otros miembros.
- Para organizar jerárquicamente los peers entre si, permitiendo asignar roles diferentes a cada grupo.

2.5. Arquitectura de CompP2P

En este apartado se entra en detalle en la arquitectura interna de la plataforma una vez introducidas las tecnologías principales sobre las que se sustenta CompP2P.
En la siguiente imagen se puede observar el diagrama del diseño basado en módulos del sistema.
CompP2P está basado en la funcionalidad clásica de un servidor. Es decir, que estará esperando permanentemente la recepción de peticiones de trabajo. Por eso mismo, el sistema se comporta como un demonio (daemon).

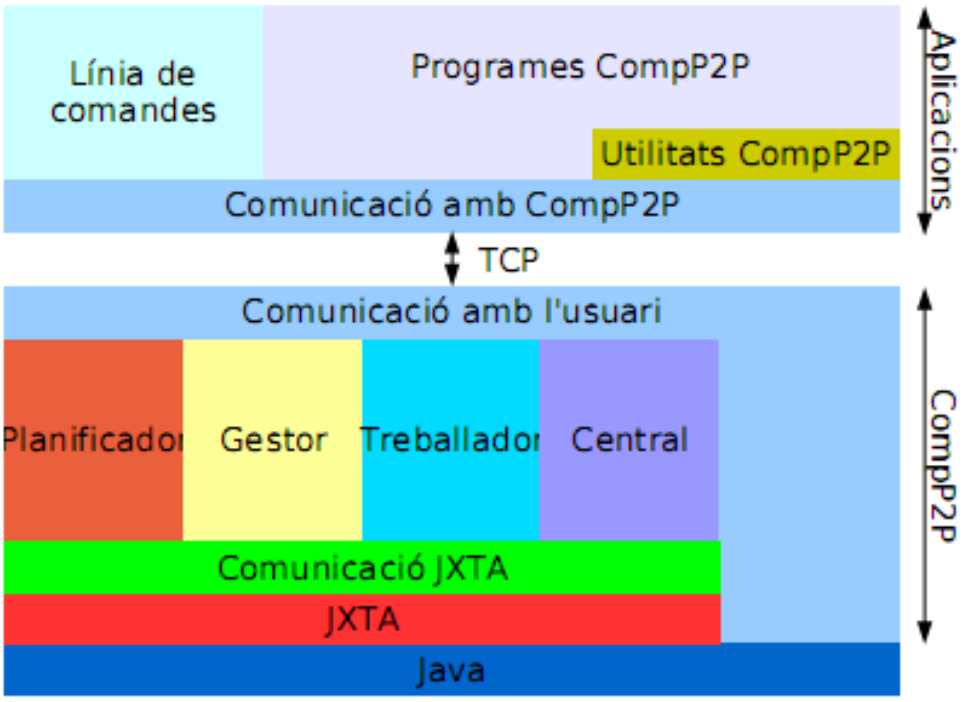


Ilustración 2: Diagrama de la arquitectura de CompP2P

2.5.1. Aplicación

En este apartado se introducirán y explicarán los mecanismos que la plataforma CompP2P implementa y proporciona para permitir la ejecución de tareas dentro del sistema, así como una introducción a la forma en que debe desarrollarse una aplicación para poder ser lanzada dentro de la plataforma.

2.5.1.1. Interfaz TCP y comunicación con el sistema

La plataforma incluye un interfaz de comunicaciones basado en sockets que permite a las aplicaciones de usuario acceder a los servicios del sistema, como son el lanzamiento de tareas o la solicitud de información del estado del sistema.
Como se ha comentado, está implementada utilizando comunicaciones TCP/IP y sockets para conectar la aplicación de usuario con un daemon dentro del middleware de CompP2P, que permite a estos entablar una comunicación y compartir información y realizar ciertas acciones. Las funcionalidades principales que se pueden realizar con este interfaz son la de ejecutar trabajos en el sistema, obtener información del estado del sistema, administración y de forma general proporcionar acceso a los recursos y servicios del sistema. Siguiendo la filosofía de software multiplataforma, el mecanismo de comunicación escogido son los sockets TCP.
A través de este módulo externo de comunicación, el demonio podrá recibir tareas y solicitud de estadísticas. El puerto TCP que se utiliza es el 35557.
La idea clave es que CompP2P se comunica con los nodos mediante JXTA, y mediante TCP lo hace con el usuario.

La comunicación con el usuario se realiza a través del puerto TCP 35557, y se puede establecer conexión con el demonio de varias formas:

- Usando uno de los dos clientes TCP que se incluyen con el paquete de la plataforma
- Conectando con el demonio directamente sobre la comunicación TCP usando alguna aplicación que lo permita, como *nc (netcat)*.

A continuación se enumeran los diferentes comandos que soporta en su versión original el

cliente TCP usando uno de los dos clientes que incorpora:

- **get:** Obtener datos del sistema.
- **set:** Modificar datos concretos del sistema.
- **job 'PATH' HASH:** Ejecutar un trabajo.
- **shutdown:** Parar el sistema CompP2P.
- **help:** Muestra un listado con los comandos soportados.
- **exit**: Salir del cliente del sistema.

Los dos comandos *principales get y set* soportan argumentos, como por ejemplo la linea *get manager peers* devuelve el numero de peers de un área. O por ejemplo *get peer*, la cual nos devolverá información sobre nuestro propio peer.
Al escribir este último comando, el sistema nos devolvería algo del aspecto de:

```
PeerName: pc1
WorkerPeerID:
urn:jxta:uuid-59616261646162614A78746150325033BAE831CABA834078926883CE7433956
903
PipeWorkerID: urn:jxta:uuid-
C61C35673DE24306A37B994723DA95B7453935CCBE714AEDADE9FAC37C80812E04
AreaPeerID:
urn:jxta:uuid-59616261646162614A78746150325033BAE831CABA834078926883CE7433956
903
PipeAreaID:
urn:jxta:uuid-61A5BF707CA646779B85E30069F800056B90D5F6B337407AAD79E1676176D0C
804
FreeMemory: 342872
ExecutedJobs: 28
```

En la plataforma se hace un uso extenso de *threads*, ya que esto permite que al mismo tiempo que se ejecutan faenas proporcionadas por el usuario, el poder recibir mensajes, o realizar cualquier otro tipo de tareas.
Esto también permite que se puedan aprovechar las capacidad de los últimos procesadores, como el Hyperthreading o los procesadores con múltiples núcleos.

2.5.1.2. Desarrollo de aplicaciones para ejecutarlas en la plataforma

El módulo *Parallel* es el encargado de encapsular los objetos que se han de ejecutar remotamente, además de servir como base para las aplicaciones diseñadas para la plataforma CompP2P. Sirve también para servir como capa superior de abstracción para comunicarse con el demonio de TCP, sin exigir el usar las funciones y sistemas de bajo nivel.

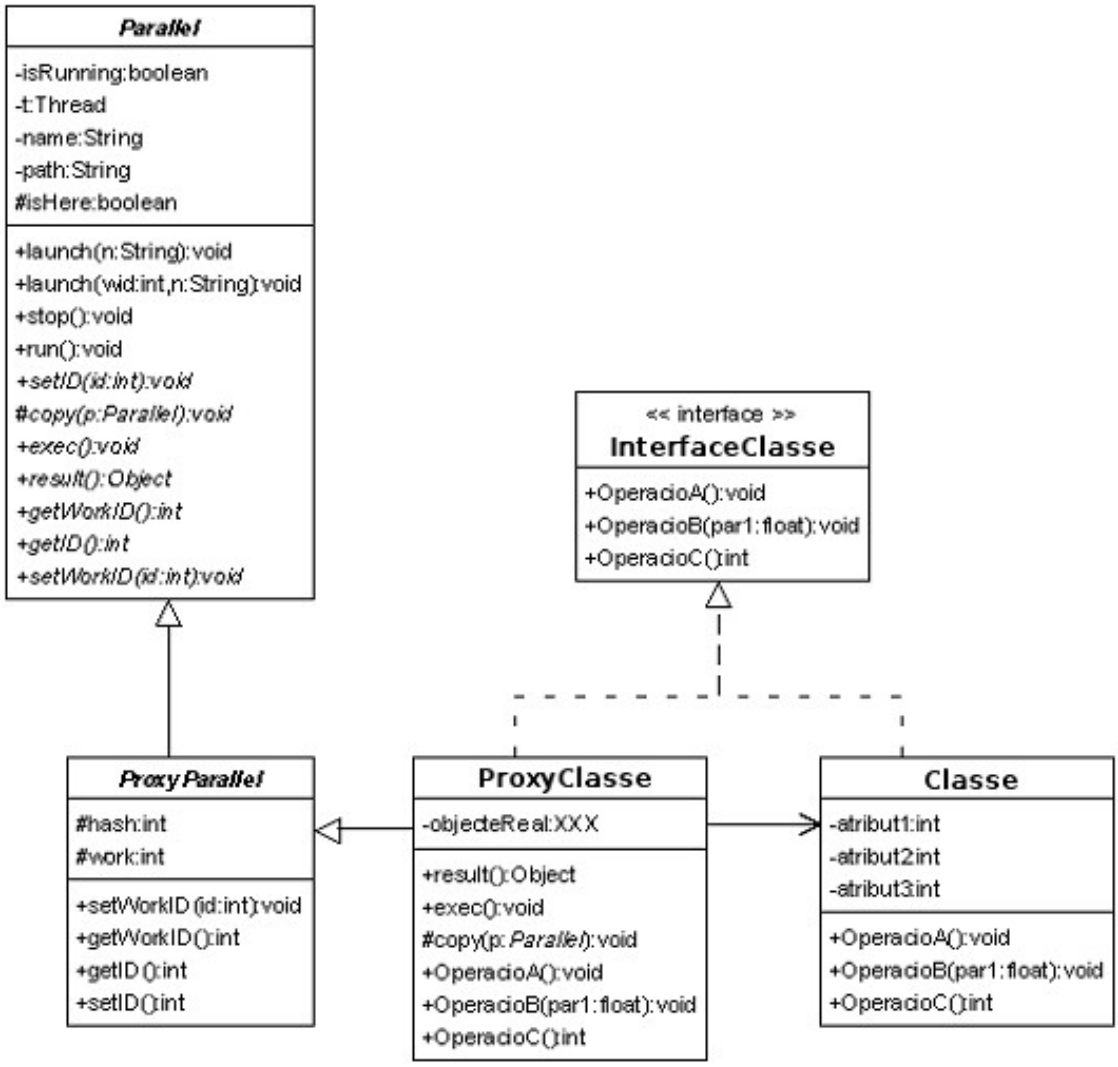


Ilustración 3: Diagrama de las clases principales

Los componentes principales son los siguientes:

- **Parallel:** Implementa los métodos que se usan para serializar, enviar, ejecutar y recibir el trabajo entre los nodos, dentro del objeto “Clase”.
- **ProxyParallel:** Sirve como emboltorio para el transporte de atributos de la clase Parallel, ya que esta no se puede serializar ni enviar directamente entre los peers.
- **InterfaceClase:** Define los métodos de “Clase” y de su proxy.
- **Clase:** En esta clase se implementa la clase principal, con todas sus funciones y atributos.
- **ProxyClase:** Esta clase facilita la ejecución remota del objeto “Clase”. Implementa los métodos: “result()”, “exec()”, y “copy()”.

A continuación se detallarán los aspectos de implementación y utilización de estos componentes:

La clase Parallel:

Como se ha dicho, esta clase implementa los métodos que se usan para serializar, enviar, ejecutar y recibir el trabajo entre los nodos dentro del objeto “Clase”.
Esta clase la entenderemos como una *superclase*, ya que todas las clases que definamos en la aplicación que deseamos ejecutar remotamente dependerán de esta.
Entre esta clase *Parallel* y la clase principal de la aplicación tendremos dos clases *Proxy* que se encargarán de implementar los métodos abstractos que tiene declarados la clase *Parallel*, y así un encapsulado de la clase de la aplicación.
También es necesario definir un *interface* que contendrá los métodos que tendrán comunes la aplicación y su *proxy*.

A continuación se detallan las operaciones que implementa la clase Parallel y que son heredadas por las aplicaciones que han de ser ejecutadas remotamente en CompP2P.

- **public Parallel()**
  Constructor de la clase

- **public void launch(String n) throws RemoteException**
  Lanza la otra operación launch(wid, archivo.jar) con el identificador de trabajo wid fijado a 0. *archivo.jar* es el archivo que contiene los archivos .jar de la clase de Parallel. Ejemplo de uso:
  ```
  try{ ejemplo.launch(archivo.jar); }
  ```
  ejemplo es alguna de estas subclases de Parallel.
- **public void launch(int widm, String n) throws** RemoteException
  Lanza la *operación run*() usando un thread de *Java. wid* es un identificador de flujo de trabajo, *y archivo.jar* es el archivo que contiene los archivos .jar de la clase de Parallel.Ejemplo de uso:
  ```
  try{ ejemplo.launch(wid, archivo.jar); }
  ```
  ejemplo es alguna de estas subclases de Parallel.
- **public void stop() throws RemoteException**
  El objeto *ejemplo* ha llamado anteriormente a la operación *launch(...)* y ya está desbloqueado, preparado para lanzar otras operaciones. Ejemplo de uso:
  ```
  try{ resultado = ejemplo.stop(); }
  ```
- **public void run()**
  Este metodo define el cómputo que se tiene que ejecutar remotamente, será el código que se ejecutará el thread que arranca el método launch. Hace lo mismo que launch(...) pero sin usar threads. Ejemplos de uso:
  ```
  ejemplo.run();
  ```
  o usando un thread:
  ```
  th = new Thread(ejemplo); th.start();
  ```
- **public void setID(int id)**
  Operación abstracta para asignar el id del archivo de la clase corresponiente al objeto distribuido.
- **public void getID()**
  Operación abstracta para consultar el id del archivo de la clase corresponiente al objeto distribuido.
- **public void setWorkID(int id)**
  Operación abstracta para asignar el ID del trabajo.
- **public void getWorkID()**
  Operación abstracta para consultar el ID del trabajo.

- **abstract public void exec()**
  Clase abstracta.
- **abstract public Object result() throws RemoteException**
  Clase abstracta consultora.
- **abstract protected void copy(Parallel p)**
  Clase abstracta copiadora.

La clase ProxyParallel:

Es una clase auxiliar de la clase Parallel, que forma parte de esta. Su función es la de dotar a la clase Parallel con los mecanismos necesarios enviar ciertos valores a otros nodos (modificar: permite serializar y distribuir los trabajos entre los nodos).
Implementa la clase *Serializable*,

A continuación se detallan las operaciones que implementa la clase ProxyParallel.

- **public void setID(int id)**
  id contiene el identificador hash del archivo .jar. Ésta operación actualiza el identificador del arhivo. Ejemplo de uso:
  ```
  ejemplo.setID();
  ```
- **public int getID()**
  Obtiene el identificador hash del archivo .jar. Ejemplo de uso:
  ```
  int hashID = ejemplo.getID();
  ```
- **public void setWorkID(int id)**
  id contiene el identificador hash del trabajo. Ésta operación actualiza el identificador del trabajo. Ejemplo de uso:
  ```
  ejemplo.setWorkID();
  ```
- **public int getWorkID()**
  Obtiene el identificador hash del trabajo. Ejemplo de uso:
  ```
  int hashID = ejemplo.getWorkID();
  ```

Para adaptar un objeto distribuido a la plataforma CompP2P, y que pueda ser ejecutado en la misma, hay que tener en consideración siguientes puntos:

1. En el objeto base
   - El objeto base ha de ser ejecutado remotamente, por lo que toda la información que pueda necesitar le ha de ser proporcionada como argumento al crearlo, ya que no se puede interaccionar con él por linea de comandos una vez ha empezado su ejecución.
   - Se debe implementar el método *copia*.
   - La clase tendrá que disponer las operaciones *calculate()* y *result()*.
2. En cuanto al proxy del objeto:
   - Se tendrá que implementar el método *calculate()* para que esta implemente la *interface* de la aplicación, como en el objeto base.
   - Como en el objeto base, hay que implementar el método *copy()*, además de los métodos *exec()* y *result()*.
3. Interficie del objeto
   - Hay que declarar los métodos *calculate()* y *result()*.

2.5.1.3. Lanzando trabajos en la plataforma

Para poder ejecutar trabajos dentro del sistema se han de seguir unos pasos y cumplir una serie de requisitos.
Lo primero que hay que recordar es que CompP2P dispone de un interfaz de red funcionando como demonio TCP al cual se le pueden enviar mensajes y recibir respuestas. Es a través de este interfaz de red al que enviaremos las tareas que queramos que se ejecuten.

En el siguiente diagrama se detallan los pasos por los que debe pasar una tarea hasta ser ejecutada en el sistema.

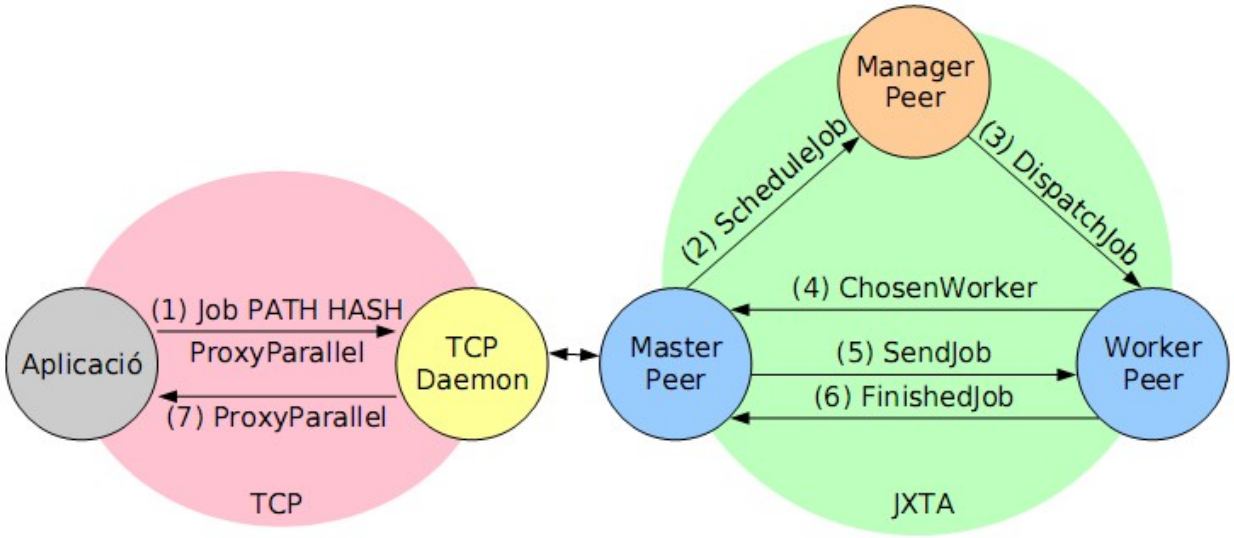


Ilustración 4: Ejecución de un trabajo en ComP2P

A continuación se detallan los pasos mostrados en la anterior ilustración.

1. La aplicación ha de mandar el mensaje "*job "PATH HASH""* al daemon TCP de la plataforma, donde PATH es la ruta a la tarea, y HASH es un código único de hash generado a partir de la tarea. Mira si tiene el código fuente de la tarea descomprimido en su directorio de clases, y si lo tiene, espera a recibir un objeto serializado de Parallel lo pasa al módulo principal. Una vez recibido el objeto en el módulo principal, crea un objeto *job* con todos los datos de la tarea, y es el mismo que será almacenado en la cola de tareas. El peer es marcado como master de la tarea.
2. Cuando el master tiene la tarea, le envía la información de la misma a su gestor de área.
3. Cuando el gestor tiene la tarea, se la envía a un peer que lo llamará worker, y si no encuentra ninguno, delegará la tarea a otros gestores de área.
4. El trabajador que reciba la tarea informará al master de que él la va a realizar.
5. El master enviará el objeto de la clase ProxyParallel serializado al trabajador para que lo ejecute.
6. El master se pone en espera para recibir los resultados de la ejecución. Hace uso

de un semáforo para evitar esperas activas. Cuando el trabajador devuelve la tarea al master, éste la guarda en su cola de tareas, que no es más que una cola FIFO .

7. El trabajo es devuelto al demonio TCP una vez lo ha recibido el módulo principal.

2.5.1.4. Aplicación de muestra SumP2P

Se seguirá un ejemplo práctico para ver como se adapta un objeto a CompP2P. El ejemplo que se utiliza en el proyecto de final de carrera relativo a la plataforma CompP2P es el de suma distribuida. He decidido usar el mismo ya que se encuentra bien documentado y es una aplicación sencilla de analizar.

El objetivo de la aplicación es el de sumar todos los números que se encuentran entre dos números proporcionados *x* e *y* de la forma siguiente:

x + (x + 1) + (x + 2) + ... + (y − 2) + (y − 1) + y

Lo que la aplicación distribuida hará es dividir las series de números en series pequeñas, y entonces repartirlas para que sean calculadas.
A continuación se muestra la especificación de las diferentes clases implementadas en el ejemplo:

Clase *SumP2P* (Implementa las clases *InterfaceP2P* i *Serializable*):

- **SumP2P(int f, int l)**
  Constructor. El primer número tiene que ser más grande que el segundo.
- **public void calculate() throws RemoteException**
  Calcula la suma de los números entre x e y y guarda el resultado en una variable.
  Ejemplo de uso:
  ```
  try{ ejemplo.calculate(); }
  ```
- **public Object result() throws RemoteException**
  Muestra el resultado por pantalla y lo guarda en una variable. Ejemplo de uso:
  ```
  try{ result = ejemplo.result(); }
  ```

- **public void copia(SumP2P nou)**
  Operación copiadora. Ejemplo de uso (ejemplo2 es una variable de la clase SumP2P):
  ```
  ejemplo.copia(ejemplo2);
  ```

Clase *ProxySumP2P:*

- **public ProxySumP2P(int f, int l)**
  Operación constructora.
- **public void calculate() throws RemoteException**
  Calcula la suma de los números entre x e y y guarda el resultado en una variable. Ejemplo de uso:
  ```
  try{ ejemplo.calculate(); }
  ```
- **public Object result() throws RemoteException**
  Muestra el resultado por pantalla y lo guarda en una variable. Ejemplo de uso:
  ```
  try{ result = ejemplo.result(); }
  ```
- **public void exec()**
  Calcula la suma de los números entre x e y y guarda el resultado en una variable. Hace lo mismo que la *operación result*(). Ejemplo de uso:
  ```
  try{ ejemplo.exec(); }
  ```
- **protected void copia(Parallel nou)**
  Operación copiadora. Ejemplo de uso (ejemplo2 es una variable de la clase SumP2P):
  ```
  ejemplo.copia(ejemplo2);
  ```

Clase *InterfaceSumP2P*:

- **Object result() throws RemoteException**
  Operación abstracta.
- **void calculate() throws RemoteException**
  Operación abstracta.

Clase SumP2P

Es la clase principal de la aplicación de suma distribuida, y por lo tanto, es donde se implementarán realmente los métodos.

```
package sump2p;
import parallel.*;
import java.io.*;

public class SumP2P implements InterfaceSumP2P, Serializable{
  private int first;
  private int last;
  private int result;
```

```
  public SumP2P(int f, int l){
          first = f;
          last = l;
          result = 0;
  }
```

Inicializa las variables

```
  public void calculate() throws RemoteException{
          int i;
          for(i=first; i<=last; i++){
                    result=result+i;
          }
  }
```

Es un bucle desde first hasta last en el que va sumando el resultado parcial

```
  public Object result() throws RemoteException{
          return new Integer(result);
  }
```

Devuelve el resultado como un entero

```
  public void copia(SumP2P nou){
          first = nou.first;
          last = nou.last;
          result = nou.result;
  }
```

Clase ProxySumP2P

Implementa los métodos de la clase *SumP2P* y vigila que la clase no sea llamada desde otra aplicación.

```
package sump2p;
import parallel.*;
import java.io.*;

public class ProxySumP2P extends Parallel implements InterfaceSumP2P,
Serializable{
   SumP2P obj;

   public ProxySumP2P(int f, int l){
            obj = new SumP2P(f,l);
   }

   public void calculate() throws RemoteException{
            if(!isHere){throw new RemoteException();}
            else{obj.calculate();}
   }

   public Object result() throws RemoteException{
            if(!isHere){throw new RemoteException();}
            else{return obj.result();}
   }

   public void exec(){
            try{this.calculate();}
            catch(Exception e) {e.printStackTrace();}
   }

   protected void copy(Parallel nou){
            SumP2P c = ((ProxySumP2P)nou).obj;
            obj.copia(c);
   }
}
```

Clase InterfaceSumP2P

```
package sump2p;
import parallel.*;

   public interface InterfaceSumP2P{
            Object result() throws RemoteException;
            void calculate() throws RemoteException;
   }
```

Por último, una vez definidas las clases principales de la aplicación, únicamente nos resta implementar la aplicación en sí, que se encargará de enlazar con CompP2P y realizar los cálculos.

CompP2P provee otra clase llamada *CompP2PUtils*, que proporciona varios métodos para obtener información del sistema. En el ejemplo la usaremos para obtener el número de peers que forman parte del sistema, de la forma siguiente:

```
CompP2PUtils stats = new CompP2PUtils();
int peers=stats.getPeers();
```

Implementación de la aplicación:

```
import cat.udl.eps.compp2p.sump2p.ProxySumP2P;
import cat.udl.eps.compp2p.utils.CompP2PUtils;

class Aplicacion{

   public static void main(String args[]){
            CompP2PUtils stats=new CompP2PUtils();
            int peers=stats.getPeers();
            int total=100000;
            System.out.println("We have " + peers + " peers, we will do " +
   peers + " jobs.");
            ProxySumP2P example[]=new ProxySumP2P[peers];
            try{
```

```
                    for(int i=0; i<peers; i++){
                    example[i]=new ProxySumP2P(i*(total/peers)+1, ((i
  +1)*(total/peers)) );
                    example[i].launch("sump2p.jar");
                    }
                    for(int i=0; i<peers; i++){
                              example[i].stop();
                              System.out.println("Result number " + i + ": " +
  ((Integer)example[i].result()).intValue());
                    }
          }
          catch(Exception e){
                    System.out.println("ERROR!");
          }
  }
}
```

2.5.2. Middleware

En este apartado se detalla la gestión que hace la plataforma de los grupos y los gestores de grupos.

2.5.2.1. Grupos

La red es segmentada y jerarquizada, tal y como se ha comentado con anterioridad.
Como todos los peers han de poderse comunicar de forma global, estos pertenecerán al grupo de los trabajadores, teniendo que conocer cada uno de ellos el estado de los otros.
Con esta estructura de red existe el problema de las entradas redundantes, y es que si en una red tenemos 1000 nodos, existirán 1000*1000 (1 millón) de comunicaciones, implicando una sobrecarga importante en la red. Para solucionar este problema, la plataforma incorpora el concepto de *manager peer*. Éste tipo de peer será el que se encargará de conocer el estado del resto de la red.
Con esto se introduce un segundo problema, y es el de que podemos tener un único manager, añadiendo un retardo considerable a las comunicaciones del sistema y que a la larga se convertirá en el cuello de botella de todo el sistema. Para solucionarlo, CompP2P introduce los grupos de área.
Para ello se definen como gestores a un numero de peers, los cuales serán los “líderes” de su área. Estos deberán conocer y mantenerse comunicados con los otros gestores de área. Para poder hacer esto último, los gestores pertenecerán a un grupo superior, el llamado de monitores.

A continuación se puede observar un diagrama con la jerarquía de grupos del sistema.

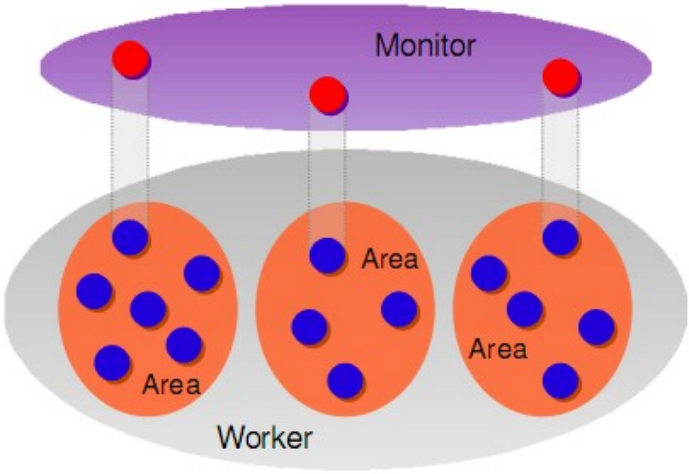


Ilustración 5: Jerarquía de grupos en CompP2P

Éstos grupos pueden ser creados estáticamente, ya que de no ser así, podría darse el caso de que los ordenadores de una sala, la cual nos interesaría dedicarlos a una tarea concreta, no pudieran encontrar el grupo a través del peer rendez-vous.

2.5.2.2. Gestores

Como se ha comentado, las áreas deben tener un gestor. Éste se encarga de conocer el estado de su área, comunicar entre sí a los nodos de la misma, y proporcionar información de su área a los otros gestores del sistema.
A parte de estas tareas de mantenimiento, el gestor tiene otra función más importante, y es la de distribuir las tareas que le llegan entre los peers de su área según unos criterios de planificación eficiente y equitativa. Para ello, el gestor dispone de un planificador, donde se almacena información de los trabajadores de su área y sobre el estado de las otras áreas del sistema.
Cuando un usuario lance una tarea, el planificador seleccionará un trabajador de su área para que la realice. En caso de saturación, el planificador enviará esta tarea a un gestor de otra área.
En la primera versión de la plataforma, el algoritmo que utiliza el planificador es *round-*

robin. Actualmente un alumno de la escuela está trabajando en mejorar y añadir funcionalidades a ésta parte de planificación de ComP2P.

A continuación se definirá el mecanismo utilizado para la entrada/salida de peers del sistema, así como la creación y destrucción de los grupos, y el mantenimiento de la jerarquía del sistema.
A continuación se muestra un diagrama de flujo sobre el funcionamiento de la jerarquía de la plataforma.

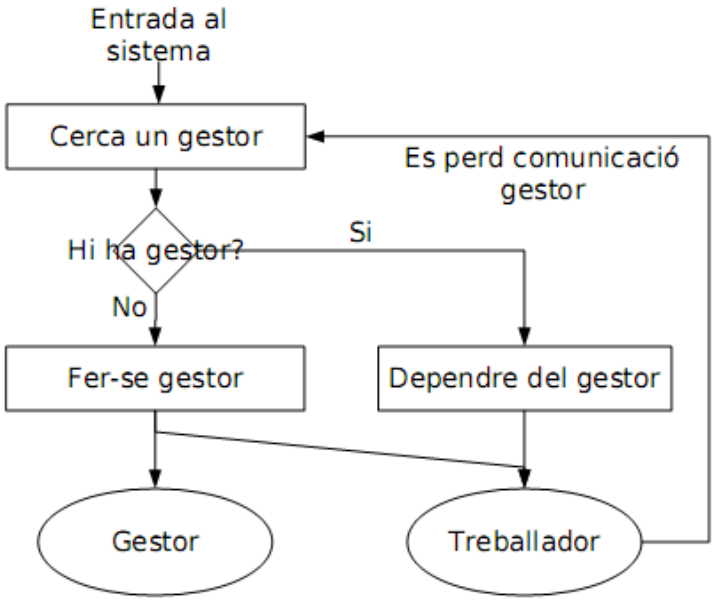


Ilustración 6: Diagrama de flujo sobre la jerarquía de la plataforma

Para entrar en el sistema, un peer puede encontrarse en tres situaciones distintas.

- La primera es que no haya ningún usuario en el sistema. En este caso, el peer creará un grupo de trabajadores y entrará a formar parte de él, ya que aún no hay ninguno. Hará lo mismo con el grupo de área. Creará un gestor en el grupo de área y se conectará a él. El gestor buscará el grupo de monitores, y lo creará y entrarña en él al no encontrarlo. El peer enviará información al gestor y esperará la recepción de tareas.
- En el segundo caso ya hay una jerarquía. Si se conecta en la misma área que el anterior, buscará un grupo de área y de trabajadores, y se unirá a él. Por último

buscará un gestor y le enviará información para poder recibir tareas.

- En el tercer caso, un usuario puede estar en otra área. En ese caso el peer se unirá al grupo de trabajadores que ya existe. Buscará un grupo de área, pero al estar en otra, creará él un grupo nuevo. Y como en el primer caso, al no encontrar un gestor de área, lo creará él. Por último le enviará información al gestor para poder recibir tareas.

En cuanto al mantenimiento de la jerarquía del sistema mientras éste está funcionando, debe llevar un control, ya que pueden darse casos en que hayan dos gestores dentro de un mismo grupo de área por ejemplo.
Para mantener la estructura de la jerarquía, los peers aprovechan los mensajes de actualización del sistema. Cada 20 segundos los peers enviará información actualizada a su gestor. A su vez, los gestores cada 20 segundos realizan comprobaciones del sistema, como el conocer el estado de los peers de su área, controlar que no entre ningún gestor en su área, y por último, proporcionar información de su área a los gestores de las otras. Con esto se consigue que los gestores conozcan el estado de toda la red, y así evitar la sobrecarga de peers o áreas.

Por último está la situación de que un nodo salga del sistema. En ésta situación se puede dar el caso que el peer que deja el sistema sea el gestor de una área. Al salir, los peers que cuelgan de él detectarán que no pueden enviar mensajes a su gestor, por lo que procederán a buscar a otro gestor para unirse al área de éste.
El otro caso es que el peer que deja el sistema sea un trabajador. El gestor de su área detectará la eventualidad, y lo eliminará de su memoria.

2.5.3. Recursos

Los recursos en la plataforma son gestionados directamente por los nodos a los que pertenecen, aunque de cara a estadísticas globales, estos recursos se suman en ese cómputo.
Los recursos principales del sistema son la potencia de cómputo (tiempo libre de CPU) y la memoria libre disponible.
La potencia de cómputo y el tiempo libre de CPU de un nodo son el utilizados para decidir si un nodo puede ejecutar una tarea o si en cambio en ese momento está ocupado y no puede tratarla.
La memoria de los nodos es gestionada directamente por la máquina virtual de Java, encargándose esta última de descargar de memoria todos aquellos objetos que no han de ser usados. También se encarga de proporcionar a la plataforma una estimación de la memoria libre que tiene el nodo, pero hay que tener en cuenta que esta memoria no se corresponde con la memoria libre total de la máquina local, sino de la memoria libre de la máquina virtual de Java.

3. Análisis de requisitos y diseño

3.1. Objetivos

El objetivo principal es el de incorporar a la plataforma CompP2P los medios necesarios para realizar sobre la misma una monitorización de su estado.

- **Distribuido y jerárquico:**

De cara a mantener una organización y estructura de las estadísticas del sistema, este debe ser jerárquico y distribuido. Es decir, que cada una de las clases o módulos de la plataforma ha de poder generar sus propias estadísticas de forma totalmente independiente del resto de módulos, además de hacerlo de una forma ordenada dependiendo de la posición que ocupe cada módulo en la estructura de las estadísticas.

- **Flexible e incremental:**

Como segunda premisa, y de cara a la escalabilidad y futura ampliación del sistema, el medio de transmisión y almacenado de las estadísticas debe ser flexible e incremental. Con ello se pretende conseguir que la incorporación de nuevos atributos o tipos de estadísticas no implique realizar cambios a nivel estructural del sistema de monitorización, además de no limitar la cantidad de información que se pueda almacenar y gestionar.

- **Separar parte funcional / parte de presentación gráfica:**

Por otra parte también es necesario el establecer desde un principio un aspecto muy importante de cara al posterior diseño e implementación del sistema de monitorización, y es el de separar de forma clara y adecuada la parte de generación de estadísticas de la parte de procesado y presentación visual de las mismas. El separar claramente la parte funcional de la visual permite un mantenimiento y ampliación de las aplicaciones a largo plazo, ya que se pueden realizar modificaciones, por ejemplo, en la forma en que se transmiten los mensajes, sin necesidad de modificar la parte encargada de representar por pantalla las estadísticas que se transmiten mediante estos.

- **Estadísticas en tiempo real:**

Otra característica muy importante a tener en cuenta es que la plataforma de monitorización permita la solicitud de estadísticas en tiempo real. De esta forma, una tercera herramienta externa podría solicitar estadísticas al sistema cada cierto periodo de tiempo, procesarlos, y mostrarlos al usuario de forma dinámica, es decir, actualizando la información cada cierto tiempo.

- **Soporte gráfico:**

De cara a proporcionar una forma más cercana al usuario para visualizar estadísticas, se marca como objetivo importante el incorporar los mecanismos necesarios para representar parte de los datos obtenidos de la plataforma CompP2P en forma de gráficas, además de permitir visualizar los datos brutos obtenidos en una forma visual clara y agradable.

3.2. Requisitos

3.2.1. Requisitos funcionales

- **Distribuido y jerárquico:**

Cada nodo de la plataforma habrá de ser capaz de generar sus estadísticas cuando le sean solicitadas, y enviarlas al solicitante, sin requerir mayor interacción, de forma que la generación global de las estadísticas sea lo más distribuida posible y con el menor retraso posible entre la solicitud de las mismas y la recepción de sus resultados.
En el proceso de cálculo de estadísticas no debe existir ningún componente centralizado.
Las estadísticas serán divididas en dominios independientes, que serán los siguientes:

- **Global**: Incluirá las estadísticas principales de todo el sistema.
- **Área**: Incluirá las estadísticas de un grupo concreto, tanto de los peers simples como del gestor de ese grupo.
- **Peer**: Incluirá únicamente las estadísticas relativas a un nodo concreto.

- **Flexible e incremental:**

La estructura de datos que se utilice como medio de almacenamiento y transmisión ha de permitir combinar nuevos datos fácilmente a un objeto ya existente, ya que al utilizarse en un medio distribuido las estadísticas de cada elemento del sistema deberán combinarse en un único objeto para ser enviadas al elemento que las solicitó.
Otro requisito esencial es que las estructuras de datos utilizadas en el almacenamiento y propagación de las estadísticas han de ser independientes de los nodos de cómputo, y por lo tanto portables. Un exemplo de estructura de dato multiplataforma flexible es XML, que de hecho, ya es utilizado en diferentes plataformas como medio de transmisión de información, como por ejemplo WSDL [WSD01] en servicios web.

- **Separar parte funcional / parte gráfica:**

La parte de funcionamiento interno de la plataforma y del sistema de monitorización ha de ser totalmente independiente de los mecanismos externos encargados de representar o procesar visualmente las estadísticas. También es necesario que el medio de almacenamiento básico de las estadísticas no esté vinculado directamente con elementos visuales ni mecanismos de representación, ya que de ser así, el realizar modificaciones a la parte visual del sistema podría implicar la realización de cambios más o menos importantes tanto en los medios de almacenamiento como en las funciones internas de la plataforma, lo cual no es deseable.

Será necesario implementar un conjunto de herramientas externas a la propia plataforma CompP2P para poder realizar ciertas tareas de presentación de estadísticas en forma gráfica, así como realizar las tareas de monitorización, ya sea mediante el uso de aplicaciones de escritorio o aplicaciones web. De esta forma se pretende separar totalmente las tareas de presentación de la propia plataforma.

- **Estadísticas en tiempo real:**

La plataforma ha de proporcionar un medio para la solicitud dinámica de estadísticas, de tal forma que puedan ser posteriormente procesadas. De cara a que esta solicitud dinámica de estadísticas no implique una penalización importante de rendimiento será necesario implementar mecanismos de cache y de ahorro de información a transmitir para evitar sobrecargar el sistema con mensajes de estadísticas. Otra forma de reducir la penalización es proporcionar los métodos necesarios para poder solicitar estadística de los aspectos que realmente sean necesarios en cada momento, sin necesidad de solicitar todo el árbol de datos completo.

- **Soporte gráfico:**

Siguiendo la premisa de separar la parte funcional de la visual, es importante que el sistema de monitorización proporcione los mecanismos y herramientas necesarias para poder representar gráficamente los datos de la plataforma. Por eso mismo, los datos numéricos como por ejemplo el número de peers del sistema, o la memoria libre habrán de poder ser representados en gráficas para poder observar el estado cambiante de todo el sistema, como de los peers y áreas por separado. Los datos que no puedan ser

representados gráficamente, como números de identificación, nombres de peers o tareas habrán de ser mostrados por pantalla mediante tablas o elementos visuales que separen claramente la jerarquía de los datos y que faciliten su lectura.

3.2.2. Requisitos tecnológicos

En este apartado detallaré y analizaré las tecnologías principales que van a ser utilizadas en el desarrollo del sistema de monitorización, así como los motivos por los que han sido seleccionadas.

3.2.2.1. XML

El *Extensible Markup Language (XML)*, es un lenguaje de programación de propósito general basado en etiquetas. Su utilidad principal es la de facilitar la transmisión de información entre sistemas diferentes, o como medio de almacenamiento de datos.
Cumple los requisitos principales de ser jerárquico, flexible e incremental.
Es jerárquico, ya que se pueden introducir datos de forma que se respete una estructura basada en la importancia de cada elemento.
Y es flexible e incremental porqué se pueden introducir nuevos datos fácilmente en elementos concretos del árbol, sin necesidad de recorrerlo entero.

A continuación se muestra un ejemplo de código XML.
El árbol de ejemplo que muestro es referente a una receta de cocina. He escogido este ejemplo porqué se puede observar muy claramente la estructura y la jerarquía de los elementos, además de incorporar atributos en los mismos.

```
<?xml version="1.0" encoding="UTF-8"?>
<recetas>
        <receta nombre=”anchoas” tiempo=”5 minutos”></receta>
                <ingredientes>
                        <ingrediente nombre=”anchoas” cantidad=”6”>
                        </ingrediente>
                        <ingrediente nombre=”sal” cantidad=”1”>
                        </ingrediente>
                </ingredientes>
        <receta nombre=”paella” tiempo=”2 horas”></receta>
```

```
                <ingredientes>
                        <ingrediente nombre=”arroz” cantidad=”4 tazas”>
                        </ingrediente>
                        <ingrediente nombre=”agua” cantidad=”1 litro”>
                        </ingrediente>
                </ingredientes>
</recetas>
```

En el entorno CompP2P he utilizado XML tanto como medio de comunicación como sistema de almacenamiento.
Se ha usado como medio de comunicación tanto para transmitir estadísticas parciales entre nodos trabajadores y monitores, y entre las aplicaciones de solicitud de estadísticas y el nodo que las genera en la plataforma.
También se ha utilizado como medio de almacenamiento en las aplicaciones que acompañan a la aplicación web CompP2PWeb, guardando en el disco duro las estadísticas generadas cada minuto y utilizadas después por la aplicación web cuando estas han de ser procesadas, ya sea para mostrar datos alfanuméricos o generar gráficas a partir de esos mismos datos.

3.2.2.2. XSLT

Extensible Stylesheet Language Transformations (XSLT) es un tipo de lenguaje de programación basado en XML y que se usa precisamente para transformar documentos XML existentes en otros nuevos, o hacer legibles para los humanos estos mismos documentos.
Generalmente las transformaciones XSLT se usan para convertir archivos con datos escritos en XML a formas que puedan ser leídas por las personas, como por ejemplo una página HTML.

A continuación se muestra un ejemplo de código de transformación XSLT:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- variables GLOBALS -->
<xsl:variable name="dir_images">graficos</xsl:variable>

<xsl:template match="recetas">
        <html>
                <head>
                        <link rel="stylesheet" type="text/css"
href="css/recetas.css"/>
                        <title>Recetas</title>
                </head>
                <body>
                        <!-- Titulos -->
                        <div class="header" align="center">
                                <b>Recetas XML</b>
                        </div>

                        <div class="subheader" align="center">
                                <p></p><p></p>
                                Recetas ordenadas por título
                        </div>

                        <!-- Crida per ordenar las recetas -->
      <div>
      <xsl:apply-templates select="receta">
                                <xsl:sort select="titol"/>
    </xsl:apply-templates>
      </div>

    </body>
  </html>
```

```
</xsl:template>
</xsl:stylesheet>
```

Con este ejemplo de transformación XSLT y el archivo XML del apartado anterior se generaría una página HTML donde las recetas de cocina serían visualizadas de forma estructurada, además de mostrar separadas cada una de sus partes y elementos gráficos.

3.2.2.3. CSS

El *Cascading Style Sheets (CSS)* es un lenguaje de hoja de estilo que se usa para especificar la representación gráfica de documentos escritos en lenguajes basados en etiquetas, como XML y HTML. El objetivo principal de usar hojas de estilo es el de poder separar en un elemento web los contenidos, que generalmente está escrito en HTML, y la representación gráfica de dicho contenido, que se correspondería con la hoja de estilos, permitiendo modificar cualquiera de las partes de forma independiente.
En nuestro caso, las hojas de estilo CSS se utilizan para especificar el aspecto visual final de las estadísticas XML ya transformadas mediante XSLT, lo que nos proporciona la posibilidad de personalizar el estilo de la página HTML generada, sin necesidad de modificar en ningún momento ni el archivo de datos XML ni el código de transformación XSLT. Una de las ventajas que aporta el usar hojas de estilo es que en cuestión de segundos se puede cambiar la apariencia de las estadísticas resultantes con el mero hecho de utilizar un archivo *CSS* distinto.
A continuación se muestra una porción de ejemplo de un código CSS:

```
.Area{
  padding: 0;
  margin: 0;
  font-family: sans-serif;
  font-size: small;
  width: 99%;
  position: relative;
  left: 12;
  border-bottom: 1px black solid;
```

```
  border-left: 1px black solid;
  border-right: 1px black solid;
  border-top: 1px black solid;
}

.Area div.AreaID{
  padding: 4px 18px 4px;
  /* background: rgb(0,51,51); */
  background: rgb(127,63,63);
  color: white;
}



.Area div.details{
  padding: 4 4 4;
}

.Area div.title{
  padding: 4 4 4;
  font-size: large;
  font-style: oblique;
}
```

Como se puede observar, el código escrito en CSS está distribuido en bloques separados con la sintaxis siguiente:

```
.Nombre_bloque{
        atributos
}
```

Los bloques nos sirven para marcar los diferentes “grupos” de estilos que vamos a definir. Por ejemplo si quisiéramos especificar los estilos para un mensaje de correo electrónico por ejemplo definiríamos los grupos siguientes:

- Título
- Detalles

- Mensaje
- Firma

Luego para cada bloque especificaríamos unos atributos especiales para diferenciarlos los unos de los otros, como por ejemplo al bloque de título usando un tipo de letra más grande y un color distinto, y para los detalles un tipo de letra más pequeño con un fondo gris claro.
Además para cada bloque podemos definir sub-bloques, es decir, hacer que un bloque pueda tener áreas con estilos diferentes que a su vez dependan jerárquicamente del estilo del bloque superior.
En el código CSS de ejemplo se puede ver esta situación con el bloque “Area”, del que dependen varios sub-bloques, como “title” o “details”.
Para aprovechar éste código CSS hay que llamarlo desde un contenido escrito en HTML. A continuación se muestra un código HTML en el que se usan varias zonas dibujables “ ” en las que se especifica que debe usarse un elemento del archivo CSS, mediante la propiedad “class”:

```
<div class="Area">
<div class="AreaID">
<b>AreaID:
</b>urn:jxta:uuid-59616261646162614A78746150325033009 5F6BECAC64B88945FF15
8EE70FDD503</div>
<p>
</div>
```

El resultado final, una vez aplicado el estilo CSS es el siguiente:

AreaID: urn:jxta:uuid-59616261646162614A78746150325033009 5F6BECAC64B88945FF158EE70FDD503

Ilustración 7: Elemento HTML representado usando estilos CSS

Si no se hubiese usado un estilo CSS, o si este estuviese deshabilitado, el resultado sería el de un simple texto plano:

AreaID: urn:jxta:uuid-59616261646162614A78746150325033009 5F6BECAC64B88945FF158EE70FDD503

Ilustración 8: Elemento HTML representado sin usar estilos CSS

3.2.2.4. SWT

El *Standard Widget Toolkit (SWT)* es un conjunto de componentes *Java* desarrollados por el proyecto *Eclipse* para la construcción de interfaces gráficas. SWT fue creado con el propósito de ofrecer al desarrollador unos componentes de GUI nativos para cada sistema operativo, a diferencia de otras alternativas *como Swing el cual está codificado enteramente en Java*.

Gracias a esta naturaleza de componentes nativos, el rendimiento es superior, pero se paga un precio en el aspecto relativo a la apariencia gráfica, ya que en cada sistema operativo las aplicaciones desarrolladas con SWT se visualizarán de forma distinta, adaptada al entorno gráfico del sistema operativo en el que se esté ejecutando la aplicación.

Ejemplo de código escrito con SWT para dibujar por pantalla un botón con texto e imagen:
package org.eclipse.swt.snippets;

```
/*
 * Button example snippet: a Button with text and image
 *
 * For a list of all SWT example snippets see
 * http://www.eclipse.org/swt/snippets/
 *
 * @since 3.2
 */
import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class Snippet206 {

public static void main(String[] args) {
        Display display = new Display();
        Image image = display.getSystemImage(SWT.ICON_QUESTION);
        Shell shell = new Shell(display);
        shell.setLayout (new GridLayout());
        Button button = new Button(shell, SWT.PUSH);
        button.setImage(image);
        button.setText("Button");
        shell.setSize(300, 300);
        shell.open();
        while (!shell.isDisposed ()) {
                if (!display.readAndDispatch ()) display.sleep ();
```

```
        }
        display.dispose ();
}
}
```

Su filosofía de utilización es la de crear un objeto por cada elemento visual de la ventana gráfica, y establecer para cada uno de ellos las propiedades adecuadas, como tamaño, posición, color ...etc.
La respuesta de cada objeto se hace mediante eventos. Un evento al ser activado, ejecutará los métodos que tenga asociados.

A continuación se puede observar una captura del entorno de desarrollo *Eclipse*, el interfaz del cual está desarrollado usando los componentes *SWT*.

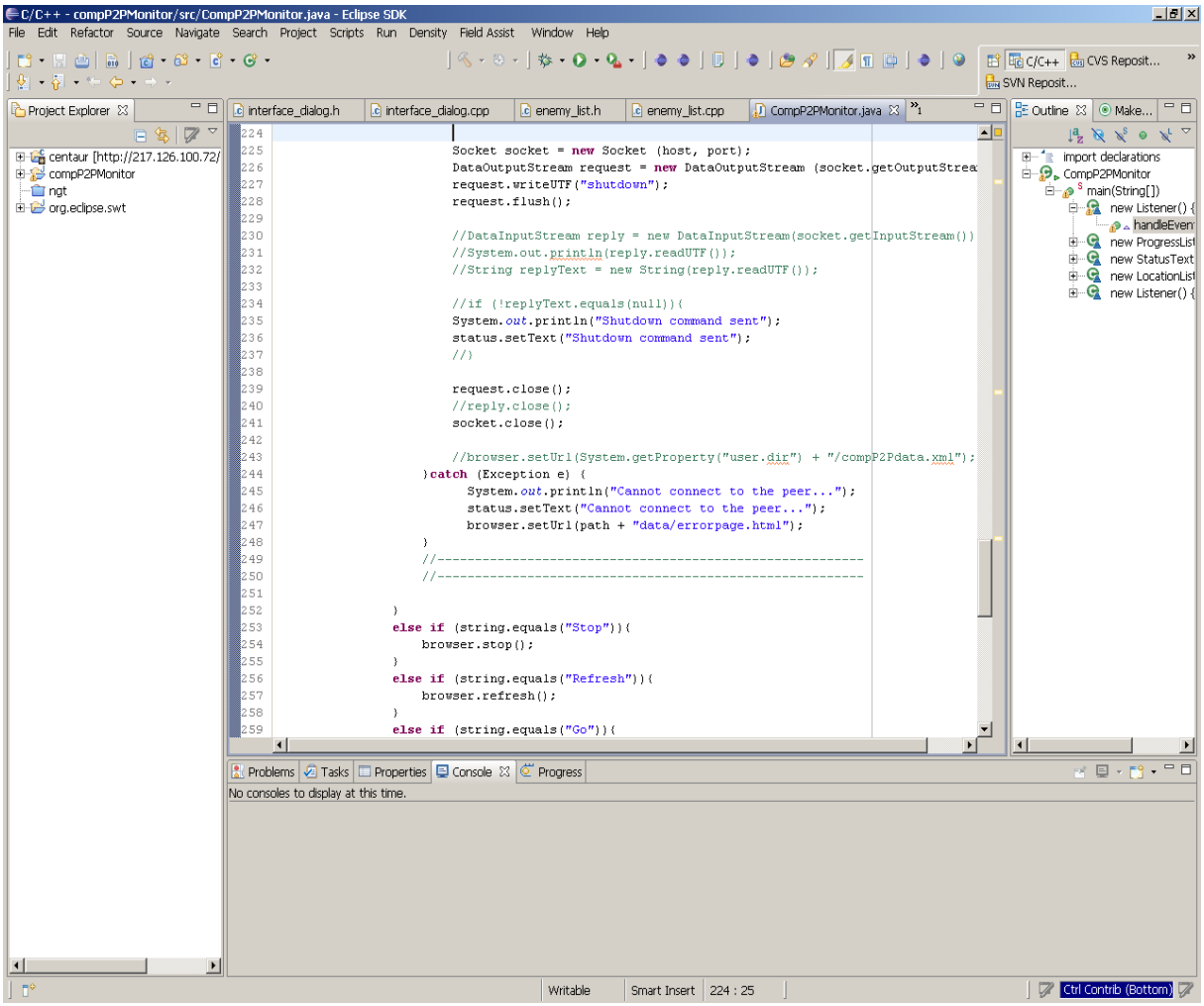


Ilustración 9: Captura del entorno Eclipse, mostrando su interfaz SWT

SWT ha sido usado en el proyecto para la implementación de una herramienta de monitorización básica, llamada *CompP2PMonitor* y que será detallada en los siguientes capítulos, que acompañará a la plataforma.
El motivo principal por el que fue seleccionado este kit gráfico y no otros como podrían ser ***QT*** *(http://trolltech.com/products/qt)* o ***GTK*** *(http://www.gtk.org)* es porqué funciona sobre Java, por lo que no nos limita tanto las plataformas de hardware y de software en las que puede funcionar.

3.2.2.5. PHP

PHP es un lenguaje de programación libre, orientado a objetos y específico para desarrollar páginas web. Al igual que **JAVA**, éste es también un lenguaje interpretado, pero algo distinto. El intérprete lee el código, lo procesa y convierte los resultados a código ***HTML***. El intérprete **PHP** está muy optimizado en las últimas versiones y se encuentran disponibles conectores para los servidores web más utilizados, tales como *Apache* o *Internet Information Server*, y para los más importantes sistemas operativos. Entre sus ventajas destacan su gran semejanza con C/C++, el ser un lenguaje muy conciso que no necesita un API excesivamente extensa, su libertad a la hora de trabajar con variables ya que no es obligatorio el especificar el tipo de dato de los mismos, y la existencia de complementos al intérprete, como son funciones de trabajo con PDFs, conectores a bases de datos como *PostgreSQL* o *MySQL* , tratamiento de imágenes y gráficos, mecanismos de cifrado de datos ...etc. Otro añadido interesante al lenguaje es un paquete de funciones llamado *PEAR* que se instalan en el propio servidor PHP y que amplían en gran medida el API del mismo.

El motivo de seleccionar este lenguaje es que su implantación en servidores web es muy grande, además de que permite lanzar aplicaciones del sistema o aplicaciones java, lo que nos permite separar en módulos sencillos todo el proceso de solicitud y procesado de estadísticas.

3.3 Diseño

Una vez definidos los requisitos, objetivos y funcionalidades principales paso a estudiar las soluciones para cada uno de los requisitos y objetivos planteados a lo largo de éste capítulo.

3.3.1. Estructuras de datos

Tal y como se ha detallado en los apartados de objetivos y requisitos, se ha utilizado el lenguaje XML. Siguiendo los objetivos de definir una jerarquía, en la estructura XML se han separado claramente los dominios principales: global, área, y nodo.
Los elementos principales de la estructura de datos pueden tener atributos. Para garantizar la flexibilidad de las estadísticas, los atributos son realmente elementos de XML con dos atributos llamados “name” y “value”. De esta manera, al no tratarse de atributos de elemento sino elementos operando como atributos posibilita que se puedan incorporar nuevos atributos a los elementos del sistema sin necesidad de modificar los diferentes parsers ni procesadores de datos de la plataforma, ayudando a la ampliación y mantenibilidad de la misma. Esto también permite que las diferentes aplicaciones de procesado y recolección de estadísticas que se implementen puedan tratar un numero indeterminado de atributos de forma transparente y genérica, sin necesidad de conocer los contenidos o la forma de los mismos.
En cuanto los diferentes dominios, en el global se almacenarán las estadísticas numéricas de todo el sistema, que podrán ser utilizados para generar gráficas y ser visualizadas por pantalla.
La raíz del árbol podrá contener un numero indefinido de áreas, que a su vez tendrán un listado de nodos y un elemento llamado ManagerInfo, que contendrá los datos relativos al monitor de esa área.
A su vez,cada peer tendrá los elementos PeerInfo con las estadísticas de ese nodo, y el elemento QueueJobs, que contendrá un listado de las tareas lanzadas por el nodo.

A continuación se muestra un diagrama con la estructura que se ha definido para almacenar en XML las estadísticas del sistema. Los elementos XML básicos están coloreados en azul mientras que sus atributos, que no hay que confundirlos con los atributos CompP2P “Attribute” los están en color amarillo pálido.

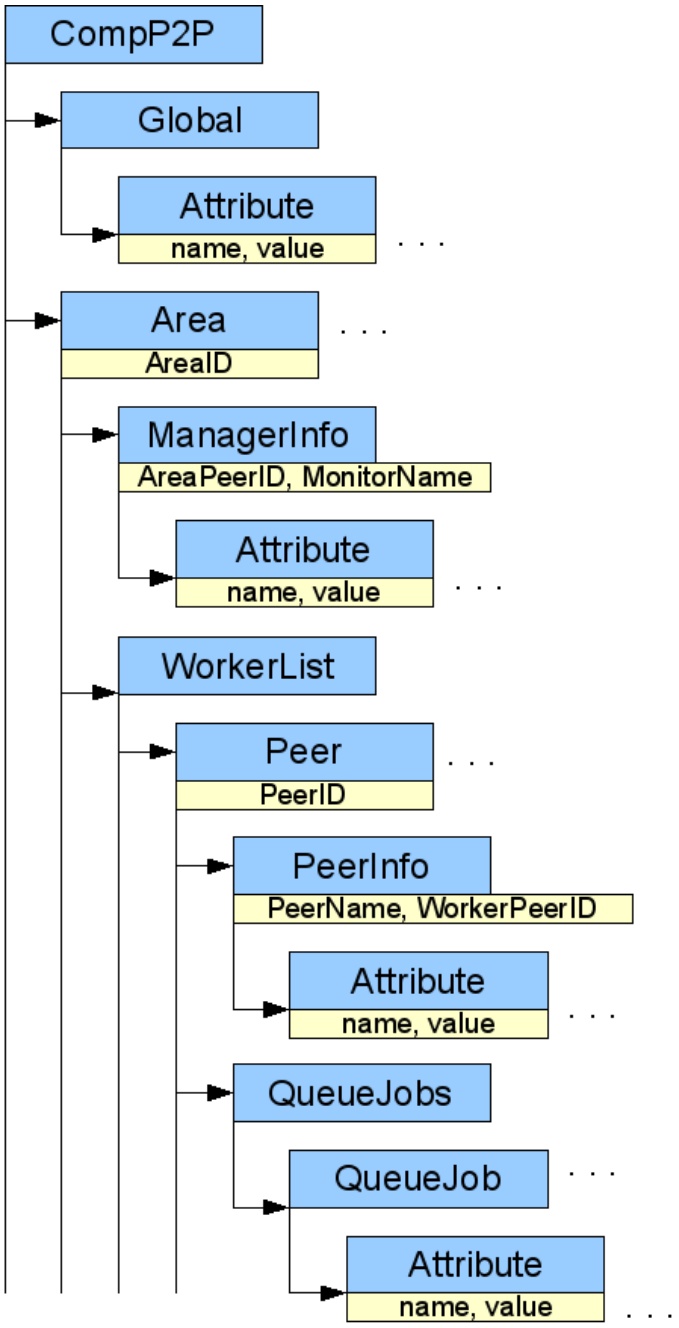


Ilustración 10: Diagrama de la estructura de estadísticas XML

Para visualizar la aplicación directa de esta estructura en un archivo XML real, a continuación se incluye un archivo de muestra generado automáticamente por la plataforma durante el periodo de realización de pruebas de funcionamiento.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<CompP2P>
    <Area
AreaID="urn:jxta:uuid-59616261646162614A787461503250335A27AB672376
4CAAAA879655EA99D49E03">
        <ManagerInfo
AreaPeerID="urn:jxta:uuid-59616261646162614A787461503250335A27AB67
23764CAAAA879655EA99D49E03" MonitorName="hyperion">
            <Attribute name="MonitorName"
value="hyperion"/><Attribute name="AreaPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250335A27AB6723764
CAAAA879655EA99D49E03"/>
            <Attribute name="PipeAreaID" value="urn:jxta:uuid-
F1BA18EE3415466F880EE3C41F3B38248F200FCAB7154C85B26AB68C1574F97B04
"/>
            <Attribute name="MonitorPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250335A27AB6723764
CAAAA879655EA99D49E03"/>
            <Attribute name="PipeMonitorID"
value="urn:jxta:uuid-3C3D4170551448F181E4E6D86918FABE395B2FCC43F14
6748B3FA5265D2CF78604"/>
            <Attribute name="NumberOfPeers" value="2"/>
            <Attribute name="NumberOfManagers" value="0"/>
            <Attribute name="GroupFreeMemory" value="2783904"/>
            <Attribute name="GroupExecutedJobs" value="0"/>
            <Attribute name="DispatchedJobs" value="0"/>
        </ManagerInfo>
        <WorkerList>
            <Peer
PeerID="urn:jxta:uuid-59616261646162614A787461503250335A27AB672376
4CAAAA879655EA99D49E03">
                <PeerInfo PeerName="hyperion"
WorkerPeerID="urn:jxta:uuid-59616261646162614A787461503250335A27AB
6723764CAAAA879655EA99D49E03">
                    <Attribute name="PeerName"
value="hyperion"/><Attribute name="WorkerPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250335A27AB6723764
CAAAA879655EA99D49E03"/>
                    <Attribute name="PipeWorkerID"
value="urn:jxta:uuid-
E193870286E147B6AF3CB12F6B978C5D1FA2544FE394408D8892B60AF1914A2704
"/>
```

```
                    <Attribute name="AreaPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250335A27AB6723764
CAAAA879655EA99D49E03"/>
                    <Attribute name="PipeAreaID"
value="urn:jxta:uuid-
F1BA18EE3415466F880EE3C41F3B3824E01661E0141A4E76866CD36FD64B94B404
"/>
                    <Attribute name="FreeMemory" value="1099112"/>
                    <Attribute name="ExecutedJobs" value="0"/>
                </PeerInfo>
                <QueueJobs/>
            </Peer>
            <Peer
PeerID="urn:jxta:uuid-59616261646162614A78746150325033009 5F6BECAC6
4B88945FF158EE70FDD503">
                <PeerInfo PeerName="caladan"
WorkerPeerID="urn:jxta:uuid-59616261646162614A787461503250330095F6
BECAC64B88945FF158EE70FDD503">
                    <Attribute name="PeerName"
value="caladan"/><Attribute name="WorkerPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250330095F6BECAC64
B88945FF158EE70FDD503"/>
                    <Attribute name="PipeWorkerID"
value="urn:jxta:uuid-
E193870286E147B6AF3CB12F6B978C5D0B97291C85E341EE8CECC757C04DC92D04
"/>
                    <Attribute name="AreaPeerID"
value="urn:jxta:uuid-59616261646162614A787461503250330095F6BECAC64
B88945FF158EE70FDD503"/>
                    <Attribute name="PipeAreaID"
value="urn:jxta:uuid-
F1BA18EE3415466F880EE3C41F3B3824D5285C0FA884412DA9C0EB45C029B55904
"/>
                    <Attribute name="FreeMemory" value="1684792"/>
                    <Attribute name="ExecutedJobs" value="0"/>
                </PeerInfo>
                <QueueJobs/>
            </Peer>
        </WorkerList>
    </Area>
    <Global>
        <Attribute name="FreeMemory" value="2783904"/><Attribute
name="NumberOfPeers" value="2"/>
        <Attribute name="NumberOfManagers" value="0"/>
        <Attribute name="ExecutedJobs" value="0"/>
    </Global>
</CompP2P>
```

Para el procesado de los árboles XML de estadísticas se implementará una clase en Java que proporcione al desarrollador los métodos necesarios para navegar entre sus elementos, extraer partes de su estructura, y realizar transformaciones sobre el árbol completo.

Ésta clase deberá cumplir con los objetivos y requisitos establecidos en este mismo capítulo, además de reducir en la medida de lo posible las penalizaciones de rendimiento que puedan implicar el recorrer árboles completos de XML.

3.3.2. Propagación

La propagación se refiere a la forma en que el sistema reacciona al recibir una solicitud de estadísticas.

Explicaré por etapas los diferentes eventos que tienen lugar cuando un nodo local realiza una solicitud de estadísticas a la plataforma.

- En una máquina, el nodo local recibe a través de su interfaz TCP la solicitud de estadísticas. Éste redirige la solicitud al nodo monitor del que depende.
- Cuando el monitor de ese área recibe el mensaje de solicitud de estadísticas de uno de sus nodos, procede a enviar un mensaje de solicitud a los nodos que de penden de él, incluido el primero que le redirigió el mensaje de solicitud. Después de esto, envía un mensaje de solicitud a los otros nodos monitor que conoce.
- Los monitores de las áreas conocidas, al recibir el mensaje realizan la misma operación que se ha detallado en el punto anterior. Una vez han generado las estadísticas, las reenvían al nodo monitor que les solicitó la información.
- Una vez el nodo monitor que ha centralizado la solicitud de estadísticas dispone de la información generada por todos los elementos del sistema, los combina en un único árbol de estadísticas y los devuelve al nodo que los solicitó al principio.
- Cuando el nodo recibe el mensaje de su monitor con las estadísticas solicitadas, las devuelve en forma de texto plano a través de su interfaz TCP a la aplicación que cursó la solicitud, el cual ya podrá procesarlas.

Para obtener una perspectiva más visual del proceso explicado anteriormente, a continuación se incluye un diagrama que detalla los diferentes pasos en la solicitud de estadísticas al sistema.

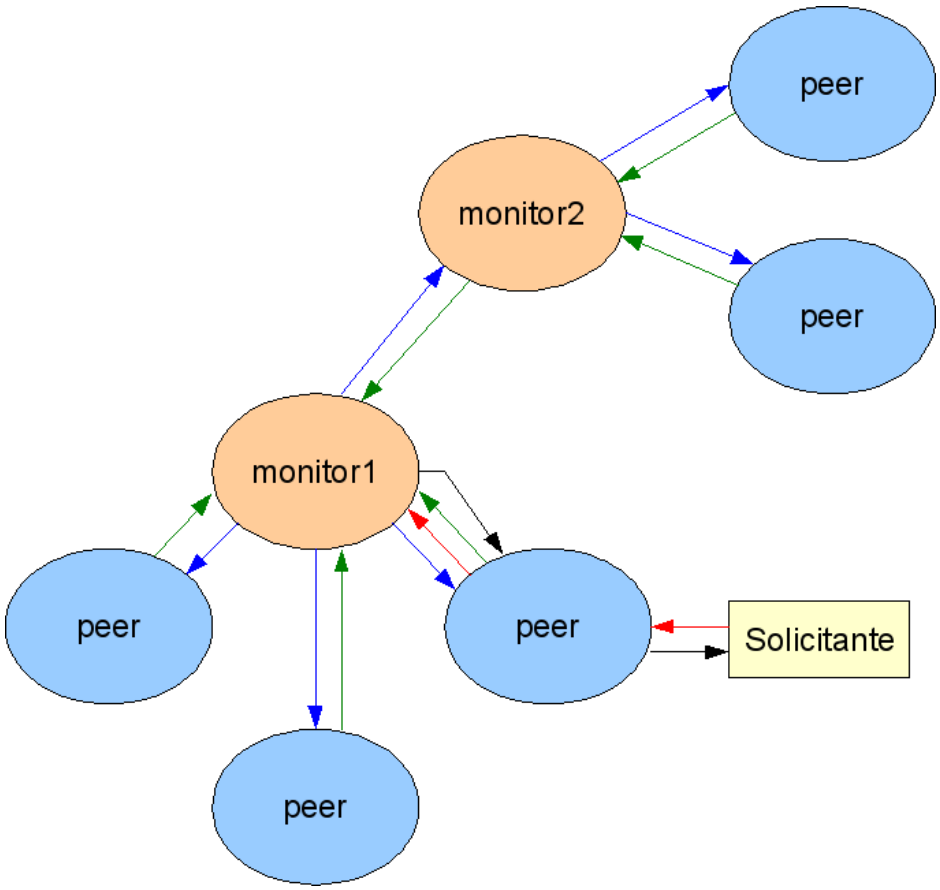


Ilustración 11: Diagrama del proceso de propagación en la solicitud de estadísticas

En el diagrama las flechas de pasos de mensajes están coloreadas para diferenciar los pasos principales. El bloque “solicitante” se refiere a la aplicación que hará la solicitud de estadísticas al nodo, que generalmente estará en la misma máquina que la aplicación.

3.3.3. Herramientas de monitorización

Una vez definidos los objetivos y requisitos del sistema de monitorización, es necesario definir la forma en que el sistema pondrá en las manos del usuario final todas sus funcionalidades.
Las herramientas principales son dos: CompP2PMonitor y CompP2PWeb. La primera permitirá monitorizar directamente el sistema a través de una aplicación gráfica clásica de escritorio. Mientras que la segunda será una aplicación web que se valdrá de dos herramientas adicionales para poder obtener las estadísticas del sistema (CompP2PResident) y procesarlas (CompP2PStatsRequester).
CompP2PResident funcionará en segundo plano realizando solicitudes de estadísticas al sistema, mientras que CompP2PWeb usará a CompP2PStatsRequester para procesar los datos que se hayan obtenido y luego mostrarlos a través de un navegador web.

En el siguiente diagrama se puede ver la forma en que estas aplicaciones enlazarán con la plataforma CompP2P.

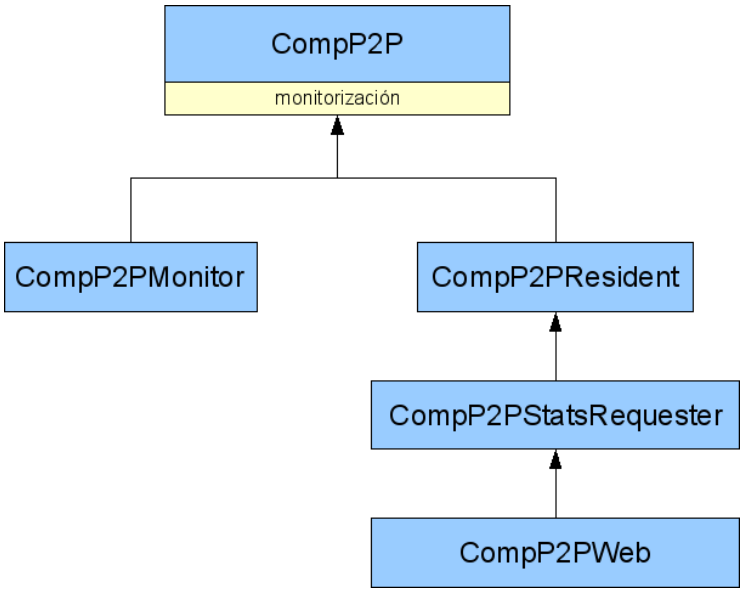


Ilustración 12: Diagrama del conjunto de herramientas de monitorización

Estas aplicaciones serán explicadas en detalle en el capítulo de implementación.

4. Implementación

En esta capítulo explicaré detalladamente los aspectos principales de implementación de cada uno de los componentes del sistema de monitorización, ya sean las modificaciones realizadas a CompP2P, las herramientas de monitorización, como la aplicación web de centralización de estadísticas.

4.1. Plataforma CompP2P

En este apartado explicaré los aspectos principales sobre la implementación de los cambios y añadidos efectuados sobre la plataforma CompP2P ya existente.

4.1.1. Propagación y solicitud de estadísticas

Las solicitudes llegan al sistema a través del interfaz TCP, que se encarga de detectar el tipo de mensaje y lanzar las funciones adecuadas.
Cuando el mensaje es de solicitud de estadísticas, el mensaje es reenviado a la clase SysJXTA, que procederá a ejecutar los siguientes pasos:

- Enviar el mensaje al monitor del que depende el nodo local.

```
        cmd = "get manager stats";
        mngMsg.setRequest();
        mngMsg.requestMsg=cmd;
        mngMsg.pipeRequester=workerPipe.getPipeID();
        mngMsg.pipeReplier=managerPipeAdv.getPipeID();
        pipeSender.management(workerGroup, mngMsg);

        while(replyMngMsg==null){
                synchronized(mngSem.get(id)){
                        try { mngSem.get(id).wait(); } catch
(Exception e) { }
                }
        }
        String aux=replyMngMsg;
        replyMngMsg=null;
```

- Una vez reciba la respuesta, la cadena de texto que contiene las estadísticas es reenviada a la clase TCP, que la enviará a través de sockets a la aplicación que solicitó los datos

```
//Create a XML tree from the aux string
managerXml = new XMLCompP2P(aux);
return managerXml.toString();
```

Volviendo al primer paso, cuando SysJXTA ha enviado el mensaje a su nodo monitor, este lo recibirá a través de la clase ManagerListener, que en función del tipo de mensaje tomará unas decisiones u otras.
Si el mensaje es de solicitud de estadísticas, se pasará la solicitud a la clase ManagerPeer, donde el monitor se encargará de realizar los siguientes pasos:

- Generar sus propias estadísticas

```
String replyMngMsg = null;

xml.addElement("Area");
xml.addAttribute("AreaID",
String.valueOf(managerinfo.getAreaID()));
// Generate ManagerInfo stats
managerinfo.generateStats(xml);

// Element to list workers
xml.addElement("WorkerList");
dispatcher.generateStats(xml);
```

- Solicitar las estadísticas de los nodos que de él dependen

```
System.out.println(dispatcher.getPeerList());
// Request statistics of connected peers
  String test2 = requestWorkerStatistics(xml);
```

- Solicitar las estadísticas de los otros monitores que conoce

```
            xml.elementUp();
            xml.elementUp();

            // Request statistics of known managers
            test2 = requestManagerStatistics(xml);
```

Cuando los monitores reciban la solicitud, procederán a realizar las mismas acciones, es decir, solicitar las estadísticas a los nodos que dependan de él y a los monitores que

conozca, y devolver las estadísticas al monitor que se las solicitó.

Una vez ha realizado todas estas acciones, devuelve la cadena de texto con todas las estadísticas combinadas al nodo que las solicitó.
La conversión de las estadísticas en forma de árbol XML a cadena de texto simple, y el combinado de datos de varias fuentes se realizan usando la clase XMLCompP2P, que en el siguiente apartado será explicada detalladamente.
Las funciones encargadas de enviar los mensajes de solicitud a los diferentes nodos y monitores del sistema son las siguientes:

Solicitud a un monitor:

```
  public void managerRequest(PeerGroup grp, Management mng) throws
PeerException {
    try{
      InputStreamMessageElement msgel=new
InputStreamMessageElement("Management", null, sendableInfo(mng),
null);
      if(mng.isRequest()) this.sendMessage(mng.pipeReplier, grp,
msgel);
      if(mng.isReply()) this.sendMessage(mng.pipeRequester, grp,
msgel);
    } catch(Exception e) {
      System.out.println("[Management] Error sending 'statistics'
message to a manager.");
      System.out.println(e.getMessage());
    }

  }
```

Solicitud a un nodo simple:

```
  public void workerRequest(PeerGroup grp, Management mng) throws
PeerException {
    try{
      InputStreamMessageElement msgel=new
InputStreamMessageElement("Management", null, sendableInfo(mng),
null);
      if(mng.isRequest()) this.sendMessage(mng.pipeReplier, grp,
msgel);
      if(mng.isReply()) this.sendMessage(mng.pipeRequester, grp,
msgel);
    } catch(Exception e) {
      System.out.println("Error sending 'statistics' message to a
worker.");
```

```
      System.out.println(e.getMessage());
    }
  }
```

Estas funciones usan el API de JXTA y de CompP2P para el paso de mensajes.
JXTA incorpora varias formas de enviar mensajes. Por defecto se ha usado el envio de mensajes simple, ya que los otros métodos no eran interesantes e nuestra situación ya que al realizar los envios con control de recepción el tiempo de espera entre el envío y la recepción de la respuesta era demasiado alto.

4.1.2. Generación de estadísticas

Para cumplir los objetivos de jerarquía y de distribución, cada clase principal incorpora un método llamado generateStats(XMLCompP2P xml), que se encarga de la generación interna de las estadísticas de esa clase, así como de su combinación en el árbol XML de datos que se le proporciona.

Tomaremos como ejemplo la generación de estadísticas de la clase PeerInfo, que como su nombre indica, se encarga de gestionar los datos principales de un nodo simple.

```
public void generateStats(XMLCompP2P xml){

    // Add each attribute

    xml.addElement("PeerInfo");
    xml.addAttribute("PeerName", String.valueOf(peerName));
    xml.addAttribute("WorkerPeerID", String.valueOf(workerID));
    xml.addP2PAttribute("PeerName", String.valueOf(peerName));
    xml.elementUp();
    xml.addP2PAttribute("WorkerPeerID", String.valueOf(workerID));
    xml.elementUp();
    xml.addP2PAttribute("PipeWorkerID",
        String.valueOf(workerPipe));
    xml.elementUp();
    xml.addP2PAttribute("AreaPeerID", String.valueOf(areaID));
    xml.elementUp();
    xml.addP2PAttribute("PipeAreaID", String.valueOf(areaPipe));
    xml.elementUp();
    xml.addP2PAttribute("FreeMemory", String.valueOf(freeMemory));
    xml.elementUp();
    xml.addP2PAttribute("ExecutedJobs", String.valueOf(execJobs));
    xml.elementUp();
    xml.elementUp();

 }
```

A la función se le proporciona un árbol XML de la clase *XMLCompP2P*, que permite introducir, eliminar y modificar elementos y atributos de cualquier parte de su estructura de datos.
En este caso, PeerInfo crea un elemento con su mismo nombre, y en él introduce una

serie de atributos que se corresponden con las estadísticas principales que la clase gestiona.
Para mantener la estructura del árbol hay que navegar al elemento en el que se han introducido datos, utilizando la función *elementUp()*.

Las otras clases del sistema generan las estadísticas de la misma forma, obteniendo eso sí, los datos de las formas que éstas requieran.

Con el uso de un nombre de función única, e incluyéndola en todas las clases se consigue una gran flexibilidad e independencia entre clases, ya que cada una por si sola es capaz de generar sus estadísticas sin dedicar atención a lo que realicen las otras.

4.2. XMLCompP2P

4.2.1. La clase XMLCompP2P

XMLCompP2P es una clase que he escrito en Java para facilitar las tareas relativas al procesamiento de los árboles XML de estadísticas que se utilizan en los diferentes componentes de monitorización de la plataforma.
La clase sirve como capa de abstracción sobre los diferentes APIs de procesamiento de objetos XML que proporciona Java, lo que ha permitido incorporar métodos y atributos especialmente diseñados para interactuar de forma más directa y sencilla con los árboles XML que se utilizan para gestionar las estadísticas.
Las funcionalidades principales son las siguientes:

- Navegación y acceso a los diferentes campos, atributos y dominios de las estadísticas.
- Añadir, modificar y eliminar, tanto atributos como elementos nuevos de estadísticas.
- Procesado gráfico del fichero XML, permitiendo convertir el árbol en una página HTML, utilizando filtros de estilo XSLT y CSS.
- Conversión y almacenamiento del árbol completo de estadísticas a un archivo de texto plano, que puede ser abierto y procesado con cualquier software que soporte XML.

En el siguiente apartado se detallarán los métodos y atributos que incorpora la clase para poder utilizar estas funcionalidades.
También es importante destacar que para la implementación de la clase se han utilizado exclusivamente clases y objetos estándares de Java para la interacción con XML, como son los parsers *w3c, sax, y java.xml,* permitiendo que la clase pueda funcionar en cualquier máquina virtual java sin necesidad de instalar librerias ni clases adicionales.

4.2.2. Métodos de la clase

- **public XMLCompP2P(String inputStr)**

Constructor que crea un árbol de estadísticas a partir de una cadena de texto que contenga el código de un árbol XML.

- **public XMLCompP2P(Document doc)**

Constructor que crea un árbol de estadísticas a partir de un objeto de árbol XML de Java del tipo *Document*.

- **public void readFromFile(String filename)**

En un árbol de estadísticas ya creado, aunque vacío, introduce los datos a partir de un archivo XML proporcionado por el argumento de la función.

- **public void addElement(String tagName)**

Añade un elemento XML en el elemento actual. El nombre del elemento a crear se proporciona por el argumento de la función.

- **public void addP2PAttribute(String name, String value)**

Añade un atributo al elemento actual. Los atributos P2P son diferentes a los atributos estándares de XML. Los primeros son realmente un subelemento llamado “Attribute” que contiene los atributos “name” y “value”, que serán inicializados con los respectivos argumentos de la función.

- **String getP2PAttribute(String attribName)**

Obtiene el valor del atributo de tipo P2P con el nombre proporcionado como argumento de la función.

- **String updateP2PAttribute(String attribName, String attribValue)**

Modifica el valor de un atributo P2P con el nombre y nuevo valor proporcionados. Realmente lo que hace es eliminar el atributo anterior y crear uno de nuevo con el valor

actualizado pero con el mismo nombre que el eliminado.

- **public void elementUp()**

En la jerarquía del árbol, establece como elemento actual al superior del anteriormente establecido como actual.

- **public void elementRoot()**

Establece como elemento actual al elemento raíz del árbol.

- **public void goToElement(String elementName)**

Establece como elemento actual al elemento con el nombre proporcionado como argumento de la función.

- **public String toString()**

Devuelve un objeto de tipo String que contiene el código XML correspondiente al árbol de la clase.

- **public String writeFile(String filename)**

Escribe un archivo de texto plano que incluirá el código XML correspondiente al árbol de la clase.

- **public String[] getPeerList()**

Devuelve una matriz con el listado de peers que contiene el árbol de estadísticas.

- **public String[] getAreaList()**

Devuelve una matriz con el listado de áreas que contiene el árbol de estadísticas.

- **public Document extractPeer(String peerID)**

Devuelve un objeto XML Java del tipo Document que contiene únicamente la sección del árbol correspondiente al peer con identificador proporcionado a la función.

- **public Document extractArea(String areaID)**

Devuelve un objeto XML Java del tipo Document que contiene únicamente la sección del árbol correspondiente al área con identificador proporcionado a la función.

- **public Element extractElement(String elementName)**

Devuelve un objeto XML Java del tipo Element que contiene únicamente el elemento del árbol correspondiente al identificador proporcionado.

4.2.3. Implementación de la clase

El procesado de los árboles XML se realiza principalmente usando los APIs que proporciona Java. La clase actúa como capa intermedia, por lo que realizando pocas llamadas a las funciones de la misma se evita el tener que trabajar con las APIs XML de Java, que son más complejas y amplias. Con esto se consigue que el desarrollador pueda centrarse en el procesado de los árboles XML y no en la complejidad de utilización de los pársers XML.

La parte de navegación de la clase se realiza manteniendo en memoria un apuntador al elemento con el que se está trabajando actualmente. Con las funciones de navegación lo único que se hace es cambiar el elemento al que se apunta, y realizar alguna acción adicional si ésta es requerida.
La localización de elementos concretos en el árbol se efectúa aprovechando las características jerárquicas que ofrece XML. Primero se localiza la zona del árbol en la que ha de encontrarse el elemento a buscar. Acto seguido se recorren los subelementos de esa zona hasta dar con el elemento o atributo adecuado examinando su nombre de objeto.
La conversión del árbol a cadena de texto, como el volcado del mismo a un archivo de texto se realiza mediante funciones de transformación que proporcionan los pársers de Java, por lo que se pueden ampliar las posibilidades de conversión de los árboles a otros formatos de archivo o de datos usando funciones adecuadas de transformación, como por ejemplo para introducir las estadísticas en una base de datos.

En el apéndice de códigos de implementación se incluye el código completo de la clase XMLCompP2P, en la página .

4.3. CompP2PMonitor

4.3.1. La aplicación CompP2PMonitor

CompP2PMonitor es la aplicación básica que se ha desarrollado para realizar una monitorización del sistema CompP2P desde uno de sus peers.
Ésta herramienta está pensada como complemento a CompP2P, para que el usuario que instale en su PC un cliente para la plataforma distribuida disponga automáticamente de un medio sencillo para realizar una monitorización activa, tanto del estado de su peer local como del de todo el sistema al que está conectado.
Al tratarse de una herramienta que el usuario final ha de poder ejecutar de forma directa, y a la vez poder visualizar en ella de forma gráfica una serie de datos, se optó por desarrollarla usando el conjunto de librerías SWT, que ha sido introducido en el capítulo de *Requisitos tecnológicos*.
A continuación se puede observar una captura de pantalla de la aplicación corriendo en una plataforma Linux.

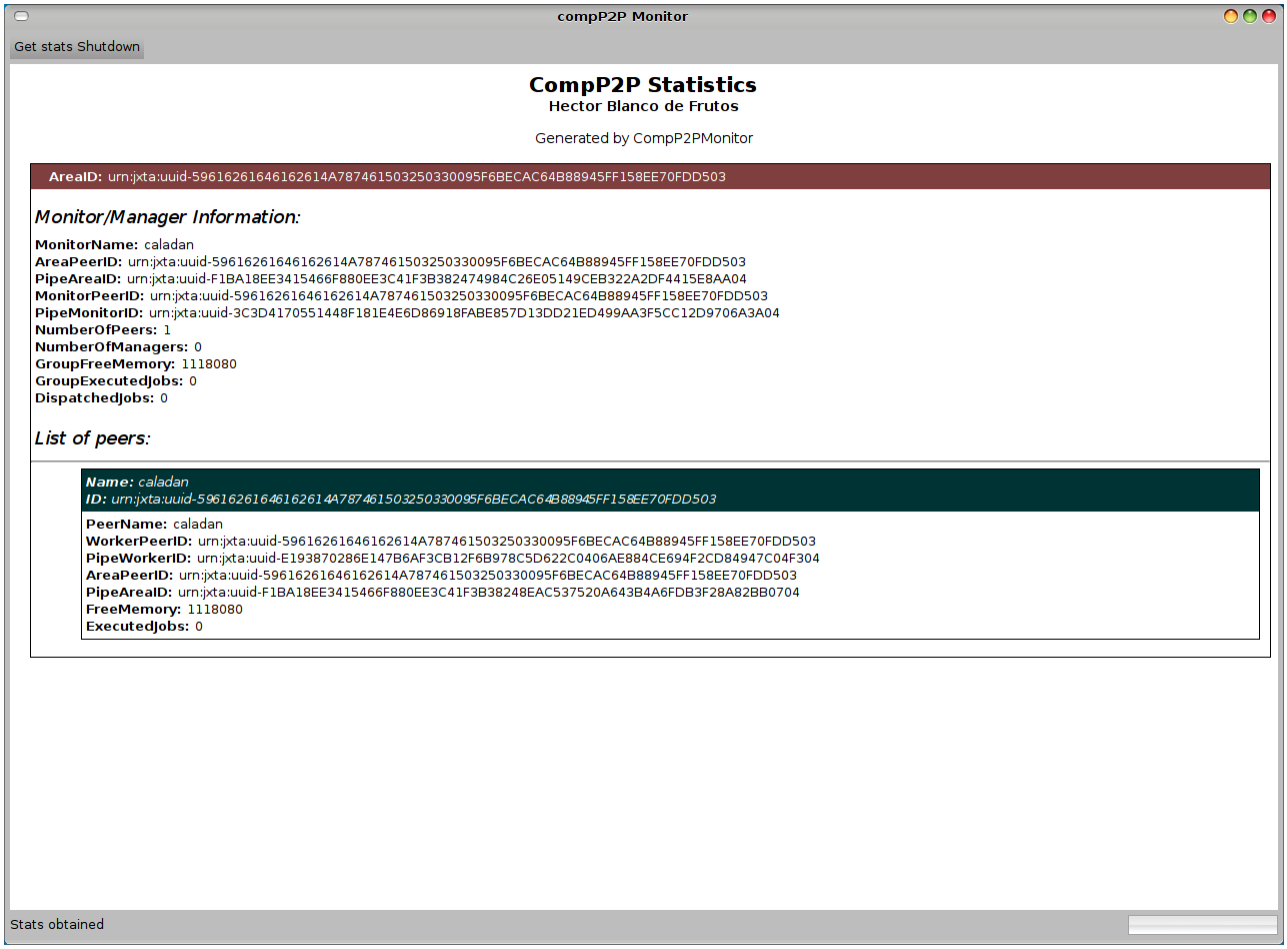


Ilustración 13: Captura de pantalla de la aplicación CompP2PMonitor

La aplicación proporciona únicamente dos botones al usuario: *Get stats* y *Shutdown.* A continuación se explica qué acciones realiza, y como las ejecuta internamente.

- **Get stats**

Obtiene las estadísticas del sistema. La aplicación monitor solicita, mediante una conexión de sockets, las estadísticas al peer que se esté ejecutando en la máquina local. Éste peer solicitará las estadísticas a su monitor y éste último propagará la solicitud por todo el sistema, devolviendo los resultados finalmente al peer local, y éste a la aplicación monitor.

La aplicación procesará las estadísticas en formato XML generando una página visualmente legible en HTML que será mostrada en el navegador web empotrado que incorpora. Éste navegador será *Microsoft Internet Explorer*, *Mozilla,* o *Safari* dependiendo

de la plataforma en que se esté ejecutando la aplicación, ya sean Microsoft Windows, Linux, o Mac OS X respectivamente. De esto se encarga de forma automática y transparente el kit SWT.

- **Shutdown**

Solicita al peer que se esté ejecutando localmente que debe cerrarse.
La aplicación monitor envía un mensaje mediante sockets al interfaz TCP del peer local con el texto de finalización “shutdown”. Cuando el peer lo reciba, procederá a detener todos sus sistemas, desconectándose de la red de cómputo.

4.3.2. Implementación de la aplicación

La característica de implementación principal de esta aplicación es el hecho de incorporar un navegador incrustado que no dependa directamente de la plataforma en la que se esté ejecutando. La incorporación de una ventana de navegador web mediante SWT se realiza con el siguiente código, que ya realiza la captura de excepciones en el caso en que no sea posible instanciar el navegador web.

```
final Browser browser;
                try {
                        browser = new Browser(shell, SWT.NONE);
                } catch (SWTError e) {
                        System.out.println("Could not instantiate
Browser: " + e.getMessage());
                        return;
                }
```

El propio kit SWT se encarga de forma automática de instanciar el navegador web adecuado a la plataforma en la que se ejecuta la aplicación, lo que posibilita que CompP2PMonitor pueda ser ejecutado en diversas plataformas sin necesidad de reescribir partes de su código. Únicamente es necesario incluir los classpath de java para la plataforma destino, que ya se incluyen con la propia aplicación.
Para facilitar la ejecución se incluyen dos scripts de arranque: *runit.bat* para plataformas Windows, y *runit-linux-firefox.sh* para plataformas Linux que dispongan del navegador web Mozilla Firefox. (la mayoría de distribuciones Linux modernas instalan este navegador por defecto).

Las acciones se ejecutan al pulsar sobre alguno de los dos botones que incluye la aplicación. Como se ha explicado en el capítulo de requisitos tecnológicos, el kit SWT gestiona el control de pulsación de los botones mediante eventos, mecanismo habitual en la mayoría de kits de diseño de interfaces gráficas.
En el cuerpo de la aplicación se realiza la comprobación de eventos de la forma siguiente:

```
Listener listener = new Listener() {
public void handleEvent(Event event) {
ToolItem item = (ToolItem)event.widget;
        String string = item.getText();
        int errorfound = 0;

        if (string.equals("Get stats")){
                // . . .
        }
// . . .
}
```

Dentro de la secuéncia de condición “if” se ejecutan las acciones adecuadas para cada botón. En el caso de obtener las estadísticas, realizar una conexión con el nodo local y solicitarle los datos para luego guardarlos en un archivo de texto. Y en el caso de cerrar el sistema, realizar una conexión con el nodo local para enviarle un mensaje indicándole que debe cerrarse.

El código completo de la aplicación se encuentra en el apéndice correspondiente a códigos de implementación, en la página .

4.4. CompP2President

4.4.1. La aplicación CompP2President

La aplicación CompP2President, desarrollada en Java, está escrita a partir del código fuente de la aplicación CompP2PMonitor para realizar en segundo plano, cada cierto tiempo y de forma indefinida la solicitud de estadísticas al sistema CompP2P a través de uno de sus peers, y guardarlas en forma de archivo XML en el disco duro.
Su utilidad principal es la de servir a la aplicación web de monitorización CompP2PWeb, que será introducida más adelante, como fuente primaria de datos.

Su funcionamiento se basa en realizar peticiones de solicitud de estadísticas al peer local, y guardarlas en un archivo XML para que posteriormente éste pueda ser procesado por otra aplicación. Cada vez que se solicitan estadísticas al sistema, la aplicación las guarda en un archivo XML diferente, usando como nombre de archivo la fecha de solicitud, lo que permite mantener en memoria física un completo histórico del estado del sistema en diferentes puntos de tiempo. Ésto último es necesario para la generación de gráficas, tal y como se detallará en la explicación de las dos últimas aplicaciones que han sido desarrolladas.
Éste proceso se realiza en segundo plano y de forma totalmente transparente al usuario, por lo que resulta de gran utilidad para ser utilizada en servidores web que no requieran de la participación activa de un usuario.

De la aplicación CompP2PMonitor se aprovecha el código encargado de realizar una conexión mediante *sockets* con el peer local. A éste código se ha añadido la funcionalidad de obtener la hora y la fecha en la que se ha realizado la solicitud de las estadísticas para usarla como nombre del archivo, y el proceso mismo de guardado del fichero XML.

4.4.2. Implementación de la aplicación

El aspecto principal a destacar sobre la implementación de esta aplicación es el cuerpo en sí de la misma, que es un bucle infinito con un temporizador al final del mismo.
En cada *paso* del bucle, la aplicación realiza una solicitud al peer local, recibe un árbol XML en forma de cadena, la convierte en un árbol CompP2P, y lo guarda en forma de archivo XML estándar. En ese momento se activa una rutina de parada.

```
Thread.sleep(30000);
```

Ésta rutina pone en reposo la aplicación durante el tiempo indicado, en este caso 30 segundos, y luego vuelve a realizar otro paso del bucle.

El código completo de la aplicación se encuentra en el apéndice sobre códigos de implementación, en la página .

4.5. CompP2PStatsRequester

4.5.1. La aplicación CompP2PStatsRequester

ComPP2PStatsRequester es una aplicación que he desarrollado en java como intermediario entre la plataforma compP2P, a través de la herramienta CompP2President que se encarga de la solicitud y recolección de estadísticas en forma de archivos XML, y la aplicación de centralización de estadísticas CompP2PWeb, que será introducida en el capítulo siguiente.
Ésta herramienta se encarga de procesar las estadísticas generadas por la plataforma CompP2P para que puedan ser utilizadas directamente por terceras aplicaciones, procesando los archivos XML de la forma adecuada a las acciones que se le especifiquen.
Las diferentes funciones que puede realizar son las siguientes:

- Generar listados, tanto de nodos como de áreas.
- Extraer estadísticas globales, de área, o de un peer concreto.
- Generar archivos de datos de un atributo concreto con un numero especificado de muestras para el dominio que se le especifique: global, un área, o un nodo. Estos archivos de datos pueden luego ser procesados directamente con la herramienta gnuplot proporcionándole un script adecuado.

Para utilizar esta aplicación como intermediario para CompP2PWeb es necesario incluirla en la carpeta java de la aplicación web, así como asignarle permisos de ejecución en el usuario que se utiliza para el servidor web. Si no se realiza esto último, el sistema operativo no permite que la aplicación pueda acceder a los archivos de datos ni las carpetas de destino.

Es necesario proporcionar una serie de argumentos para indicarle a la aplicación qué acciones concretas debe realizar. A continuación se detallan:

- **arg1**: Modo principal: stats, chart, peerlist, o arealist. En este orden, realizan lo siguiente: generar estadísticas, gráficas, listado de nodos, y listado de áreas del

sistema.

- **arg2**: Numero de muestras. Únicamente es aplicable en el caso que generemos gráficas. En otro caso se puede introducir cualquier valor.
- **arg3**: Dominio de trabajo: global, area, o peer. Definimos sobre que dominio de estadísticas vamos a trabajar.
- **arg4**: Identificador de peer o de área.
- **arg5**: En caso de generar gráficas, este argumento será necesario, y deberá contener el nombre del atributo del que se desea generar la gráfica.
- **arg6**: -p. Se utiliza para indicar al sistema que el siguiente argumento es un path. Este comando es obligatorio ya que se deja preparada la aplicación para que en un futuro pueda generar gráficas para varios atributos al mismo tiempo.
- **arg7**: La ruta de disco en la que se almacenarán los datos generados por la aplicación.

4.5.2. Implementación de la aplicación

ComPP2PStatsRequester incorpora la clase XMLCompP2P, que la utiliza para poder recorrer los árboles XML de estadísticas y poder extraer de ellos los datos que le han sido solicitados.
Lo primero que realiza al ser ejecutada es listar los archivos de datos XML, y mediante un mecanismo de cache, carga en memoria los archivos que necesita para generar las gráficas o las estadísticas que le han sido solicitadas.
Este mecanismo de cache consiste en que una vez ha procesado un grupo de de datos, guarda en disco un apuntador al último archivo tratado, para que la siguiente ocasión que deba realizar alguna acción lo haga a partir del último dato y no desde el principio. Con esto se consigue que la aplicación requiera mucho menos tiempo de proceso en sucesivas ejecuciones de la misma.

El código fuente completo de CompP2PStatsRequester se encuentra en el apéndice relativo a códigos de implementación, en la página 119.

4.6. CompP2PWeb

4.6.1. La aplicación web ComP2PWeb

CompP2PWeb es una aplicación web que he desarrollado con el lenguaje PHP con el objetivo de servir de centro de consulta de estadísticas.
Esta aplicación, instalada en un servidor web en el que localmente se está ejecutando un nodo de la plataforma CompP2P es capaz de solicitar las estadísticas de todo el sistema, procesarlas y mostrarlas tanto en forma de datos brutos como en gráficas para observar la evolución del sistema.
A continuación se muestra la gráfica referente a la memoria libre total del sistema.

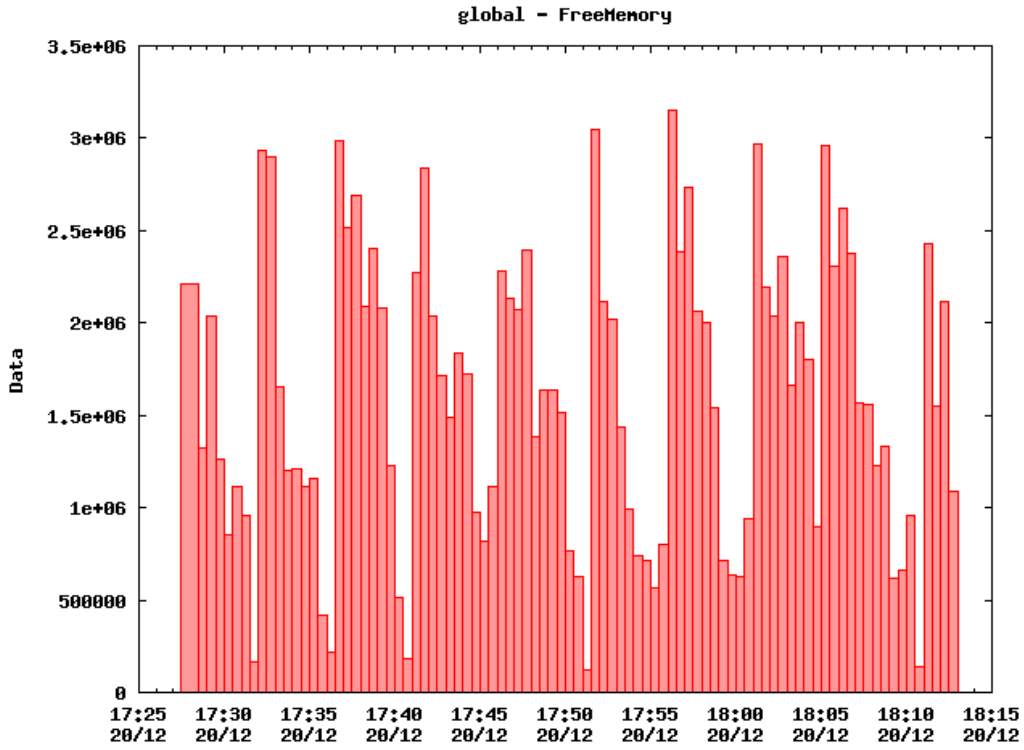


Ilustración 14: Gráfica de memoria libre del sistema

La aplicación es capaz de separar los datos y gráficas por dominios, que en nuestro caso son estadísticas globales, de área, y de nodo.
Las gráficas son generadas usando la aplicación gnuplot, por lo que es necesario tenerla

instalada en el servidor web que se vaya a utilizar como centro de estadísticas.

Para las estadísticas globales muestra gráficas numéricas, como la observada anteriormente, para los datos numéricos principales del sistema: numero de peers, memoria libre, numero de managers, y tareas ejecutadas.

Para las áreas y nodos, se muestra un listado de los que hay en el sistema, y para cada uno de ellos da la opción de visualizar sus datos brutos (View XML Statistics), o generar una gráfica de un dato en concreto con un número de muestras a elegir, a través de un formulario web estándar.

A continuación se puede observar una captura de la página de listado de nodos:

Ilustración 15: Captura del listado de peers en CompP2PWeb

A continuación se puede observar la visualización de estadísticas relativas al nodo seleccionado. El resultado es el mismo que el obtenido usando la aplicación CompP2PMonitor, ya que ambas comparten los mismos filtros de transformación XSLT y de estilo CSS introducidos anteriormente.

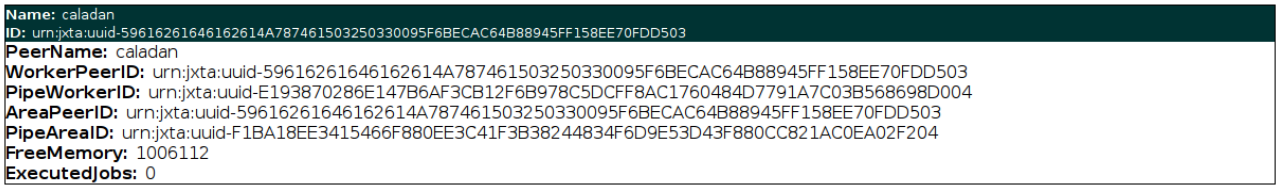


Ilustración 16: Visualización de las estadísticas de un peer en CompP2PWeb

El resultado de hacer lo mismo para un área determinada sería muy similar. Primero se muestran las estadística globales de todo el área, y luego las estadísticas locales relativas a cada nodo del área.

A continuación se puede observar el resultado de visualizar las estadísticas de un área concreta:

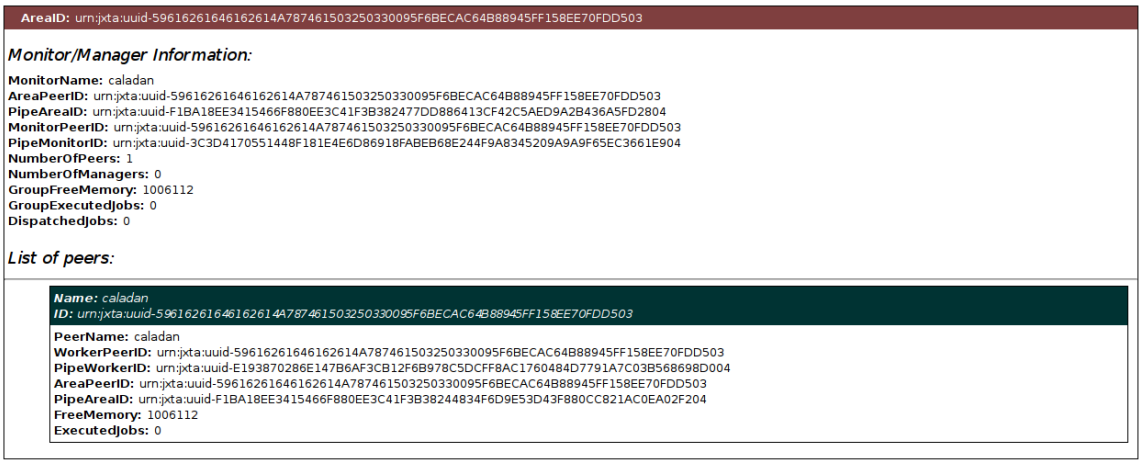


Ilustración 17: Visualización de las estadísticas de un área en CompP2PWeb

4.6.2. Implementación de la aplicación

Como se ha explicado anteriormente, la aplicación ha sido desarrollada usando el lenguaje de programación PHP.

La característica principal de la implementación de la aplicación es la forma en que procesa las estadísticas de la plataforma CompP2P, ya que no es ella misma quien lo hace, sino que hace llamadas a aplicaciones externas para llevar a cabo estas tareas.

Para la solicitud y procesado previo de las estadísticas realiza llamadas a la aplicación java CompP2PStatsRequester, que ha sido introducida anteriormente en este capítulo.

Para la tarea de generar y visualizar gráficas del sistema realiza llamadas a la herramienta gnuplot, y le proporciona tanto un script que genera la propia aplicación web como el archivo de datos generado por la aplicación CompP2PStatsRequester.

El funcionamiento de la aplicación es dinámico, por lo que dependiendo de los parámetros que el usuario le proporcione a través de los formularios de entrada, las llamadas a las aplicaciones externas serán diferentes.

Una llamada a la aplicación CompP2PStatsRequester se realiza de la siguiente forma:

```
shell_exec("java -classpath ./java:CompP2PstatsRequester.class:.
CompP2PstatsRequester stats 0 area ". $areaid ." -p ".getcwd()."
2>&1");
```

5. Resultados

En este capítulo se realizarán una serie de pruebas de funcionamiento del sistema de monitorización en un entorno de trabajo pequeño para poder comprobar su funcionamiento y respuesta.

El entorno en el que se han realizado las pruebas está compuesto por tres máquinas distintas funcionando en una red local. Se han realizado las pruebas sobre sistemas operativos distintos para poder comprobar la funcionalidad multiplataforma de la plataforma.

El entorno de pruebas se estructura de la siguiente manera:

Nodo manager del área:

- **Hyperion**
  - Intel Core 2 Duo E6400 @ 2.18 Ghz
  - 2048 MB DDR2 RAM
  - Windows XP Pro SP2

Nodos trabajadores:

- **Caladan**
  - Intel Centrino @ 1.5 Ghz
  - 1024 MB DDR RAM
  - Ubuntu Linux 7.10

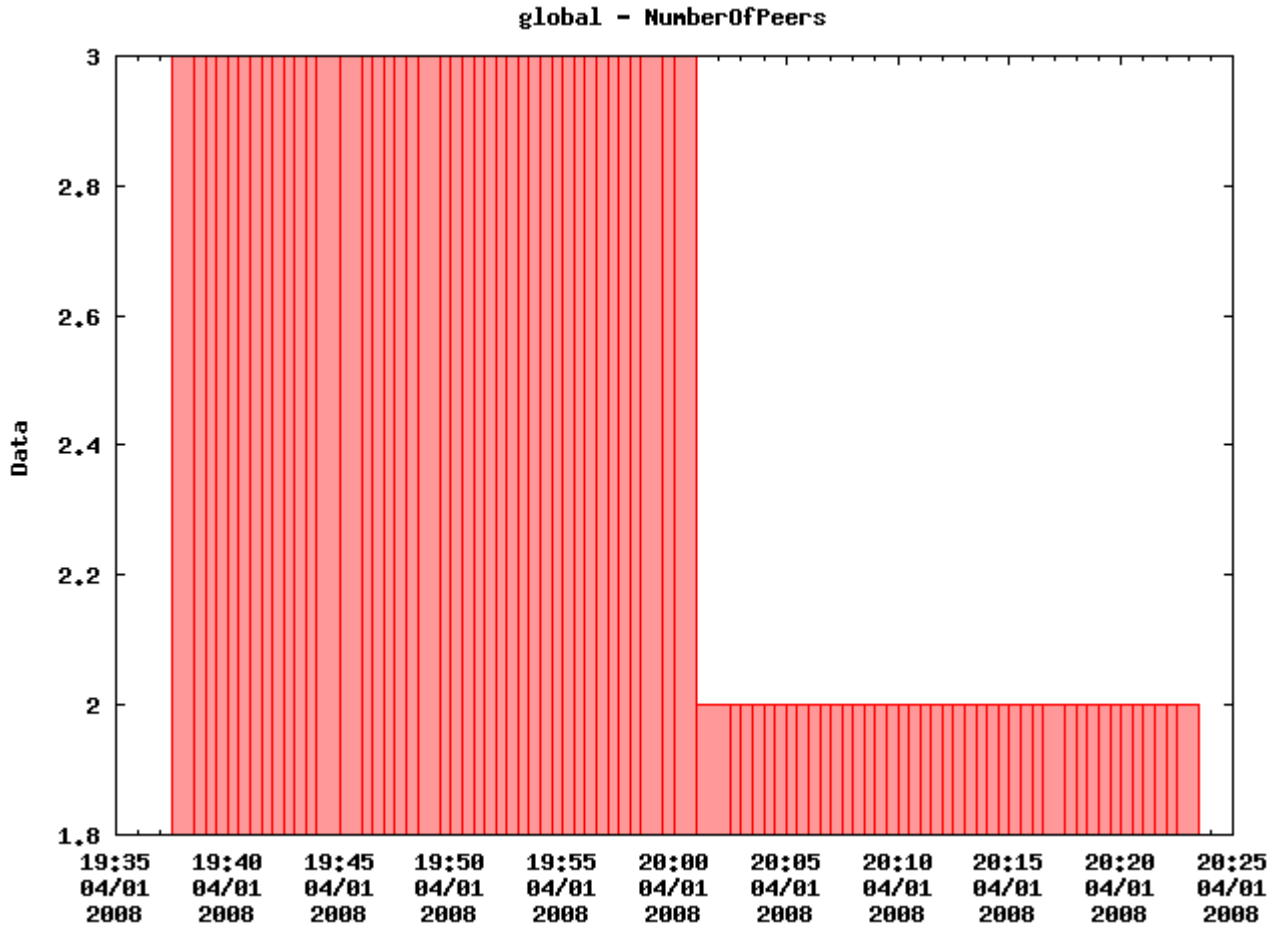
- **Dallas**
  - Intel Pentium 4 Northwood HT @ 3.0 Ghz
  - 1024 MB DDR RAM
  - Ubuntu Linux 7.10

La aplicación web y el resto de consultas de estadísticas se realizarán desde el equipo *Caladan*.

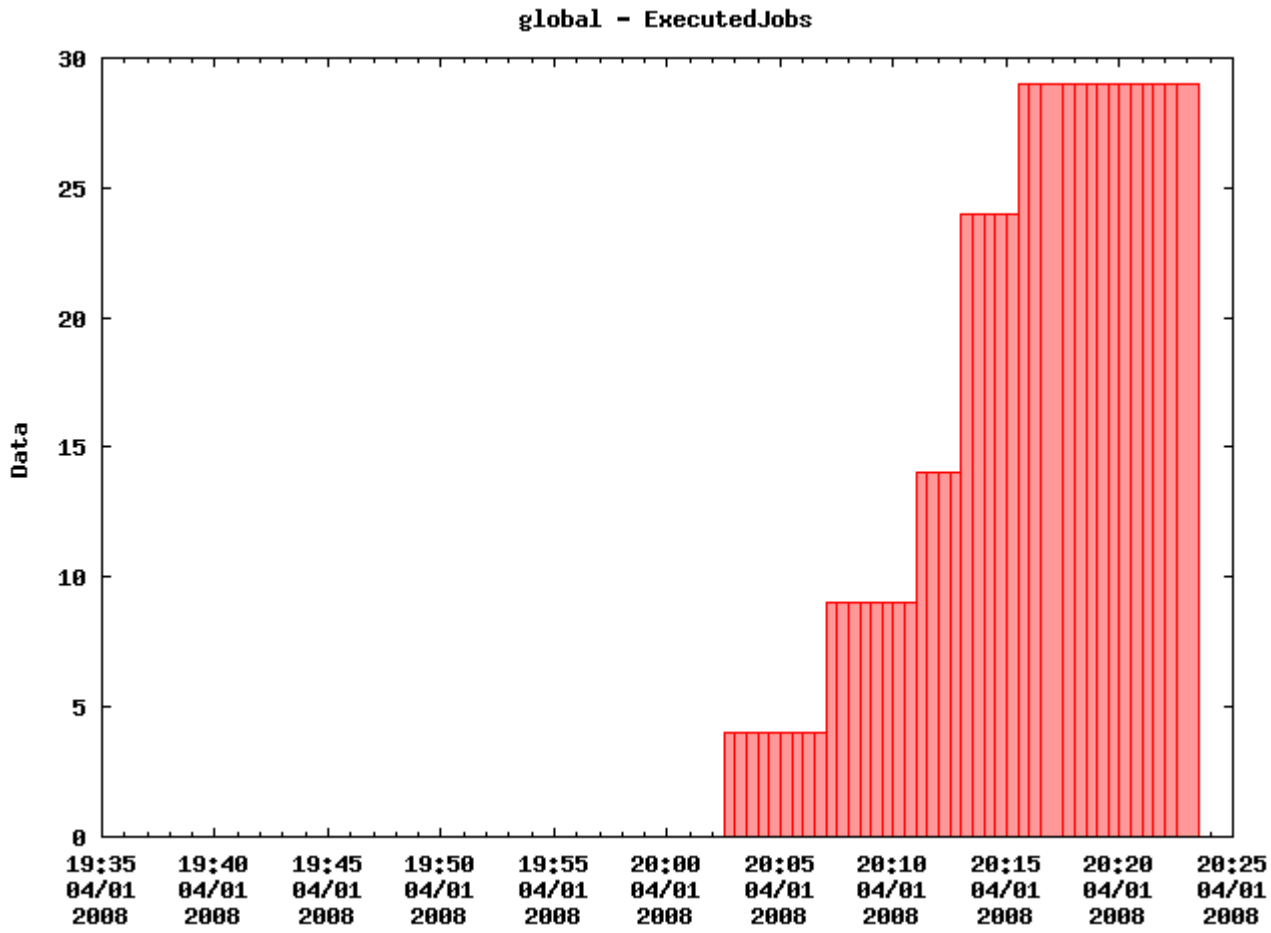
5.1. Estadísticas web

En este apartado se mostrarán las gráficas principales obtenidas del entorno de pruebas estando funcionando durante un tiempo.

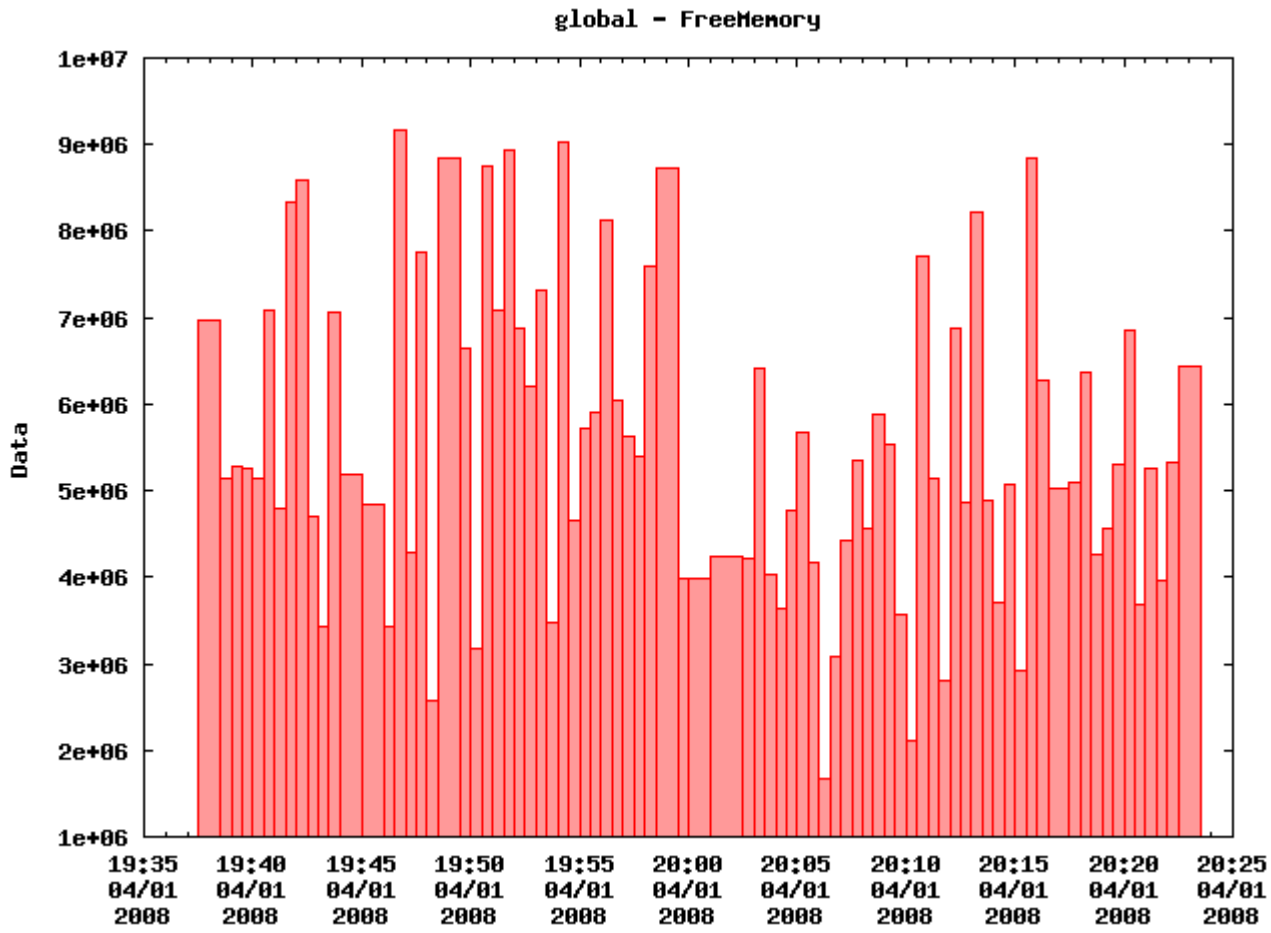
La siguiente gráfica se corresponde con el número de nodos en el sistema. Como se puede observar, al rededor de las 20:00 horas, el número desciende de 3 a 2 nodos, ya que una de las máquinas fue reiniciada. Esto nos demuestra que el sistema de monitorización es sensible a la entrada/salida de nodos en la red.



La siguiente gráfica se corresponde a la cantidad de tareas ejecutadas en el sistema. Como se puede observar, en el transcurso del tiempo, y a intérvalos diferentes han sido lanzadas nuevas tareas para comprobar el grado de monitorización del sistema.



En la siguiente gráfica se observa la cantidad de memoria libre del sistema. Ésta memoria se corresponde a la memória de la máquina virtual de java y no a la memoria física real de las máquinas.



5.2. Resultados funcionales

En este apartado se realizan varias comprobaciones y análisis de rendimiento del funcionamiento normal de la plataforma bajo diferentes situaciones de ocupación de CPU en los nodos.
El tiempo de obtención de estadísticas por parte de la aplicación residente en el caso de trabajar con 3 nodos ronda los 200 milisegundos, estando los nodos ociosos.
En el caso de estar el nodo manager ocupado, el tiempo de obtención de estadísticas es prácticamente idéntico. Comprobando los datos que muestra la herramienta *top*, se observa que la ocupación del PC apenas aumenta significativamente.
En la situación de que el nodo ocupado sea el que cursa la solicitud de estadísticas, el tiempo aumenta a 500 milisegundos.
Con estos resultados se puede llegar a la conclusión que el proceso de obtención de estadísticas no repercute notablemente en el conjunto de la red, sino que lo hace principalmente sobre el nodo que realiza la solicitud de estadísticas, ya que éste se encarga de preprocesarlas y almacenarlas en el disco.

Por otra parte, el procesado de las estadísticas en la aplicación web, que únicamente afectarán en términos de rendimiento a la máquina en la que ésta funcione, requiere un tiempo que es proporcional al número de ficheros XML que debe procesar.
El caso de estudio es el de la funcionalidad principal del primer apartado de la aplicación web: el procesado de las gráficas del estado global del sistema. Éste proceso consiste en analizar todos los archivos XML de muestra, extraer de los mismos los datos globales y generar para cada tipo de dato una gráfica para ser mostrada en el navegador web.
En el caso de la aplicación web, genera 4 gráficas globales en el entorno de 3 nodos después de haber estado funcionando durante 5 horas sin realizar ninguna actualización. Generar estas 4 gráficas en las condiciones definidas ha tomado un tiempo que ronda entre los 4000 y los 4500 milisegundos el procesar cada una de las gráficas. Hay que tener en cuenta que durante ese tiempo, el numero de archivos de datos a procesar era de 540 archivos, aunque el software de procesado limita este número a los 200 últimos ficheros. Actualizando las gráficas un instante después, este tiempo de procesado de datos se reduce a tiempos que oscilan entre los 500 y los 1000 milisegundos por gráfica.

Esto se debe al empleo de un mecanismo de cache en las estadísticas con el objetivo de evitar a la aplicación web tener que procesar todo el conjunto de datos en cada ocasión.

Debido a esta situación inherente a la arquitectura del sistema de monitorización es recomendable instalar la aplicación web en un servidor web con prestaciones suficientes, ya que un equipo doméstico de características técnicas limitadas como el utilizado es insuficiente para las tareas de procesado de datos.

En cuanto a la aplicación de escritorio de monitorización CompP2PMonitor, el tiempo requerido para procesar las estadísticas cuando se activa la función de solicitarlas es mucho inferior, ya que solamente debe recolectar los datos de un único archivo XML y visualizarlo en su navegador empotrado. Esto permite que la aplicación pueda funcionar perfectamente en cualquier ordenador doméstico o profesional sin un impacto importante en el rendimiento.

5.4. Incorporación de nuevas estadísticas

A lo largo de la memoria se ha establecido como uno de los objetivos principales del sistema de monitorización el ser flexible. En este apartado se detallará el proceso que debe seguirse para añadir una nueva estadística en el sistema.

A la hora de añadir una nueva estadística lo primero que hay que establecer es si se trata de una estadística global del sistema o únicamente de área o de peer, ya que las estadísticas globales han de ser numéricas para poder ser procesadas posteriormente de forma gráfica.

- **Estadística local (peer o área):**

Lo único que es necesario hacer es llamar al método addp2pattribute(...) sobre el árbol XML desde la clase en la que añadiremos la nueva estadística.
Por ejemplo, si queremos añadir la estadística "*CPUSpeed*" en la clase *ManagerInfo*, debemos ir al método *public void generateStats(XMLCompP2P xml)* de dicha clase.
A continuación, junto al resto de llamadas a las funciones de añadir atributo, añadiríamos la siguiente:

```
        xml.elementUp();
        xml.addP2PAttribute("CPUSpeed",
        String.valueOf(OBTENCION_VELOCIDAD));
```

OBTENCION_VELOCIDAD es el hipotético nombre que tendría una variable en la que hemos almacenado la velocidad del sistema.

- **Estadística global:**

El primer paso para añadir una estadística global, es añadirla a la clase ManagerInfo, ya que es de ésta de donde el sistema obtiene las estadísticas globales. Esto se realizaría de la misma forma en que hemos añadido una estadística en el punto anterior.
Una vez hecho esto, deberemos acceder a la clase ManagerPeer y luego a su método *public String requestManagerStatistics(XMLCompP2P xmlEl)*.
En este método, entre otras cosas, se obtiene en primera instancia las estadísticas del

manager local, y establecer sus estadísticas globales como iniciales. Para hacer esto con la nueva estadística de velocidad del sistema añadiríamos lo siguiente:

```
// CPUSpeed
xmlEl.goToElement("ManagerInfo");
tmpInt = Integer.parseInt(xmlEl.getP2PAttribute("CPUSpeed")) + 1;
xmlEl.goToElement("Global");
xmlEl.addP2PAttribute("CPUSpeed", "" + tmpInt);
```

Una vez hecho esto, deberemos obtener esta misma información de cada uno de los managers que se comunique con el local. Esto se realiza en la misma función de la clase, donde añadiríamos el siguiente código:

```
// CPUSpeed
xmlEl.goToElement("Global");
xml.goToElement("ManagerInfo");
tmpInt = Integer.parseInt(xml.getP2PAttribute("CPUSpeed")) +
Integer.parseInt(xmlEl.getP2PAttribute("CPUSpeed"));
xmlEl.updateP2PAttribute("CPUSpeed", "" + tmpInt);
```

De esta forma, la estadística nueva es obtenida de cada uno de los managers y sumada a la que ya se había introducido inicialmente del manager local.
Por último, si deseamos que esta nueva estadística global sea generada en forma de gráfica en la aplicación web, deberemos añadir el código siguiente en la sección”global” del portal:

```
generateGraphic("CPUSpeed", $files);
```

Al ser flexible el sistema de monitorización, las herramientas encargadas de procesar las estadísticas serán capaces de trabajar con los nuevos datos de forma inmediata y transparente de cara al usuario final.

6. Conclusiones

A lo largo de este documento se han introducido las tecnologías que la plataforma CompP2P tiene como base, siendo JXTA claramente la más notoria, así como parte de sus detalles internos y aspectos de implementación y especificación.
Ésto nos ha servido para entender grandes rasgos como funciona la plataforma, para así, a continuación explicar y detallar todos los puntos que han motivado la realización de este proyecto, que es el de dotar a la plataforma de un mecanismo para monitorizar su estado.
Una vez cumplido el objetivo principal de implementar los citados mecanismos, ha sido posible desarrollar una serie de aplicaciones y herramientas que permiten realizar la tarea de monitorización del sistema de distintas formas: ya sea una monitorización del estado inmediato a través de la aplicación CompP2PMonitor, o visualizar de forma gráfica el progreso de la plataforma, tanto en tareas ejecutadas como en el estado de los recursos de la misma.

En la realización de este proyecto son muchas las tecnologías utilizadas, como son Java y JXTA a un nivel de arquitectura, XML en cuanto a gestión dinámica y jerárquica de datos, PHP, XSLT y CSS en cuanto a tecnologías web, y por último SWT como tecnología de desarrollo de aplicaciones gráficas en Java para escritorio. La utilización de todas estas tecnologías y herramientas me ha aportado conocimientos muy importantes, ya que dichas tecnologías se utilizan ampliamente en infinidad de proyectos y aplicaciones de gran importancia al rededor del mundo. A parte de los conocimientos directos aportados por la utilización de estas herramientas, el surgimiento de problemas y dilemas a lo largo del desarrollo del proyecto me ha permitido el descubrir y utilizar técnicas diferentes para abordarlos correctamente, aportándome unos conocimientos de gran importancia.

La parte de resultados ha permitido detectar y observar las partes del proyecto que pueden ser mejorados para que éste realice todas las tareas con mayor soltura.
El aspecto principal en el que puede ser mejorado el sistema de monitorización es el relativo al rendimiento, ya que el uso de XML como formato de datos, pese a aportar una gran flexibilidad y jerarquía en las estadísticas, contribuye a un descenso en el rendimiento general de las aplicaciones. Una forma de mejorar los tiempos de procesado de datos, sobre todo en la aplicación web, sería el de hacer que las estadísticas fueran

procesadas directamente, al recibir datos, por la aplicación de solicitud residente en vez de hacerlo a través de una tercera aplicación como se hace ahora.

Otra alternativa sería la utilización de un formato de almacenamiento de datos distinto a XML que fuese más simple de procesar, pero que conservase la gran flexibilidad que proporciona el primero.

Las otras posibles mejoras irían encaminadas a la aplicación web, de cara a mejorar su interfaz para que fuera más amigable con el usuario, ya que la primera aproximación realizada en este proyecto tenía como objetivo el ser funcional y demostrar la viabilidad del sistema. Una vez demostrada, una siguiente versión podría enfocarse en mejorar los aspectos de interacción con el usuario mediante el uso de un interfaz más interactivo, el presentar los listados de áreas y peers de forma más clara y amigable, y la incorporación de nuevas estadísticas para ser visualizadas.

Por último, y aprovechando la infraestructura desarrollada para el sistema de monitorización, sería muy interesante la incorporación de funcionalidades de administración hasta llegar a desarrollar herramientas que permitiesen de forma gráfica gestionar el funcionamiento de todo el sistema, como añadir o eliminar tareas, cambiar preferencias de los nodos en tiempo real...etc. Con ello se conseguiría que el sistema fuese completamente interactivo de cara al usuario final.

I. Códigos de implementación

I.I. Clase XMLCompP2P

```
import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class XMLCompP2P{

		protected DocumentBuilderFactory domFactory = null;
		protected DocumentBuilder domBuilder = null;
		Document newDoc = null;
		Element rootElement= null, currentElement = null;

		Attributes attributes;

		// Basic constructor
		public XMLCompP2P()
		{
		try
		{
				domFactory = DocumentBuilderFactory.newInstance();
					domBuilder = domFactory.newDocumentBuilder();
		}

		    catch(FactoryConfigurationError exp)
		    {
		       System.err.println(exp.toString());
		    }
		    catch(ParserConfigurationException exp)
		    {
		       System.err.println(exp.toString());
		    }
		    catch(Exception exp)
		    {
		       System.err.println(exp.toString());
		    }

				newDoc = domBuilder.newDocument();

				//  Root element
				rootElement = newDoc.createElement("CompP2P");
				currentElement = rootElement;
				newDoc.appendChild(rootElement);
		}


		// Create the XML document from an existing XML string.
```

```
	public XMLCompP2P(String inputStr){
		try{

	newDoc=DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new
InputSource(new StringReader(inputStr)));
		}
		catch(Exception exp){
		System.err.println(exp.toString());
		}
	}

	public XMLCompP2P(Document doc){
		try{
				domFactory = DocumentBuilderFactory.newInstance();
					domBuilder = domFactory.newDocumentBuilder();
	  }
		catch(Exception exp){
		   System.err.println(exp.toString());
		}

		newDoc = domBuilder.newDocument();
		newDoc = doc;
	}

	public void readFromFile(String filename){

		String line;    // String that holds current file line
		String Str = new String();
	   int count = 0;  // Line number of count


		 try {
				FileReader input = new FileReader(filename);
				BufferedReader bufRead = new BufferedReader(input);



				   // Read first line
				   line = bufRead.readLine();
				   count++;

				   // Read through file one line at time. Print line # and line
				while (line != null){
					Str = Str +line;
				  //System.out.println(count+": "+line);
					line = bufRead.readLine();

				  count++;
				}
				//System.out.println(Str);

				bufRead.close();
		 }catch (ArrayIndexOutOfBoundsException e){
				// If no file was passed on the command line, this expception is
				//generated. A message indicating how to the class should be
				//called is displayed
				System.out.println("Usage: java ReadFile filename\n");

	    }catch (IOException e){
	       // If another exception is generated, print a stack trace
```

```
                    e.printStackTrace();
                }
                        // Create the XML tree using the String obtained
                        try{

            newDoc=DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new
InputSource(new StringReader(Str)));
                        }
                        catch(Exception exp)
                        {
                                System.err.println(exp.toString());
                        }
            }



            // Add a new element into the current element
            public void addElement(String tagName){
                        Element el =
(Element)currentElement.appendChild(newDoc.createElement(tagName));
                        currentElement = el;
            }

            public void addP2PAttribute(String name, String value){
              Element el = (Element)currentElement.appendChild(newDoc.createElement("Attribute"));
                        el.setAttribute("name", name);
                        el.setAttribute("value", value);
  currentElement = el;
 }

            public void addAttribute(String attribName, String attribValue){
                        //Element el =
(Element)currentElement.appendChild(newDoc.createElement(tagName));
                        currentElement.setAttribute(attribName, attribValue);
            }

            public String getAttribute(String attribName){
                        Element[] nodes;
                        String value = new String();

                        NodeList groupnodes = currentElement.getElementsByTagName("Attribute");
    int numgroups = groupnodes.getLength();
    nodes = new Element[numgroups];
    // 'For' control variable
    boolean found = false;


                        // If only one element found
              if (numgroups == 1){
                                nodes[0] = (Element)groupnodes.item(0);
                                currentElement = nodes[0];
                        //System.out.println("getAttribute: Only 1 found");
              }else if (numgroups == 0){
                        // nothing (currently)
                        //System.out.println("getAttribute: 0 found");
              }else{
                                // find the desired group element
            for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
            nodes[i] = (Element)groupnodes.item(i);
                                if (nodes[i].getAttributeNode("name").getValue().equals(attribName)){
```

```
                                        value = nodes[i].getAttributeNode("value").getValue();
                                }
                        }
                }
                return value;
        }

        public String getP2PAttribute(String attribName){
                Element[] nodes;
                NodeList groupnodes = currentElement.getElementsByTagName("Attribute");
                Element currentNode = null;
                String value = new String();

                int numgroups = groupnodes.getLength();
                nodes = new Element[numgroups];
                // 'For' control variable
                boolean found = false;

                // If only one element found
                if (numgroups == 0){
                        // nothing (currently)
                }else{
                        // find the desired attribute
                        for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                        nodes[i] = (Element)groupnodes.item(i);
                                if
(nodes[i].getAttributeNode("name").getValue().equals(attribName)){
                                                value =
nodes[i].getAttributeNode("value").getValue();
                                                        //System.out.println("getP2PAttr
ibute: found correct!!");
                                }
                        }
                }


                return value;
        }

        // Update an existing P2P Attribute with a new value
        public String updateP2PAttribute(String attribName, String attribValue){
                Element[] nodes;
                NodeList groupnodes = newDoc.getElementsByTagName("Attribute");
                Element currentNode = null;
                String value = new String();

                int numgroups = groupnodes.getLength();
                nodes = new Element[numgroups];
                // 'For' control variable
                boolean found = false;

                // If only one element found
                if (numgroups == 0){
                        // nothing (currently)
                }else{
                        // find the desired attribute
                        for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                        nodes[i] = (Element)groupnodes.item(i);
                                if
(nodes[i].getAttributeNode("name").getValue().equals(attribName)){
```

```
                                                        // remove the old value
                                                        nodes[i].removeAttribute("value");
                                                        // insert the new value
                                                        nodes[i].setAttribute("value", attribValue);

                                                }
                                        }
                                }


                                return value;
                        }

                        // NAVIGATION
                        // Go to the next upper element in the XML tree
                        public void elementUp(){
                                currentElement = (Element)currentElement.getParentNode();
                        }

                        // Go to the XML root element
                        public void elementRoot(){
                                currentElement = rootElement;
                        }

                        // Go to a specified element
                        // NOTE: Only works correctly when only one element of a name exists
                        public void goToElement(String elementName){
                         Element[] nodes;
                                NodeList groupnodes = newDoc.getElementsByTagName(elementName);
int numgroups = groupnodes.getLength();
nodes = new Element[numgroups];
// 'For' control variable
boolean found = false;

// If only one element found
if (numgroups == 1){
  nodes[0] = (Element)groupnodes.item(0);
  currentElement = nodes[0];
}else if (numgroups == 0){
  // nothing (currently)
}else{
                                        // nothing (currently)
                                        // TODO: Manage the situation of many elements with the same name
                                }
                        }

                        // Convert the XML tree to a String
                        public String toString(){
                                String str = new String();
                                //System.out.print("Converting XML tree to String...");
                                try{
                                        // Required transformations
                                        TransformerFactory tranFactory = TransformerFactory.newInstance();
                                        Transformer aTransformer = tranFactory.newTransformer();

                                        StringWriter sw=new StringWriter();

                                  Source src = new DOMSource(newDoc);
                                        StreamResult dest = new StreamResult(sw);
                            aTransformer.transform(src, dest);
```

```
                        str = sw.toString();
                        //System.out.print("  OK\n");
                }catch(Exception exp){
                        System.out.print("Converting XML tree to String...  FAIL\n");
        System.err.println(exp.toString());
                }
                //System.out.println("<-- XML Object:\n " + str + "XML Object -->");
                return str;
}

// Write the XML tree to a plain text file
public String writeFile(String filename){
                String str = new String();
                System.out.print("Writing XML file...");
                try{
                        OutputStream outStream = new FileOutputStream(filename);
                        TransformerFactory tranFactory = TransformerFactory.newInstance();
                        Transformer aTransformer = tranFactory.newTransformer();

                  Source src = new DOMSource(newDoc);
                        StreamResult dest = new StreamResult(outStream);
        aTransformer.transform(src, dest);

                        outStream.flush();
        outStream.close();
                        System.out.print("  OK\n");
                }catch(Exception exp){
                                                        System.out.print("  FAIL\n");
            System.err.println(exp.toString());
                }
                return str;
}

// Obtain a list of Peers in the XML tree
public String[] getPeerList(){
                Element[] nodes;
                NodeList groupnodes = newDoc.getElementsByTagName("Peer");
                Element currentNode = null;
                String peers[] = new String[1];

                int numgroups = groupnodes.getLength();
                nodes = new Element[numgroups];
                // 'For' control variable
                boolean found = false;

                // If only one element found
                if (numgroups == 0){
                        // nothing (currently)
                }else{
                        peers = new String[numgroups];
                        // find the desired group element
                        for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                nodes[i] = (Element)groupnodes.item(i);
                                        peers[i] = nodes[i].getAttributeNode("PeerID").getValue();
                        }
                }
                //Returns a String array with the identifiers of each Peer.
                return peers;
}
```

```
		// Obtain a list of Areas in the XML tree
		public String[] getAreaList(){
				Element[] nodes;
				NodeList groupnodes = newDoc.getElementsByTagName("ManagerInfo");
				Element currentNode = null;
				String peers[] = new String[1];

				int numgroups = groupnodes.getLength();
				nodes = new Element[numgroups];
				// 'For' control variable

				boolean found = false;
				// If only one element found
				if (numgroups == 0){
						// nothing (currently)
				}else{
						peers = new String[numgroups];
						// find the desired group element
						for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
				nodes[i] = (Element)groupnodes.item(i);
							peers[i] =
nodes[i].getAttributeNode("AreaPeerID").getValue();
						}
				}
				//Returns a String array with the identifiers of each Area.
				return peers;
		}

		// Extract the portion of a XML tree corresponing to a specified Peer
		public Document extractPeer(String peerID){
				Element[] nodes;
				NodeList groupnodes = newDoc.getElementsByTagName("Peer");
				Element currentNode = null;

				int numgroups = groupnodes.getLength();
				nodes = new Element[numgroups];
				// 'For' control variable
				boolean found = false;

				// If only one element found
				if (numgroups == 0){
						// nothing (currently)
				}else{
						// find the desired group element
						for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
				nodes[i] = (Element)groupnodes.item(i);
						if (nodes[i].getAttributeNode("PeerID").getValue().equals(peerID)){
								currentNode = (Element) nodes[i];
						}
						}
				}
				Document doc = null;

				try{
				domFactory = DocumentBuilderFactory.newInstance();
				domBuilder = domFactory.newDocumentBuilder();
		  }
				catch(Exception exp){
						System.err.println(exp.toString());
		  }
```

```
                // Create a new XML document and append the extracted portion to it
  doc = domBuilder.newDocument();
  Node tmpNode = doc.createElement("CompP2P");
  doc.appendChild(tmpNode);
                Element el = (Element)tmpNode.appendChild(doc.adoptNode(currentNode));

                return doc;
}

// Extract the portion of a XML tree corresponing to a specified Area
public Document extractArea(String areaID){
                Element[] nodes;
                NodeList groupnodes = newDoc.getElementsByTagName("Area");
                Element currentNode = null;

                int numgroups = groupnodes.getLength();
                nodes = new Element[numgroups];
                // 'For' control variable
                boolean found = false;

                // If only one element found
                if (numgroups == 0){
                                // nothing (currently)
                }else{
                                // find the desired group element
                                for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                nodes[i] = (Element)groupnodes.item(i);
                                if (nodes[i].getAttributeNode("AreaID").getValue().equals(areaID)){
                                                currentNode = (Element) nodes[i];
                                }
                                }
                }

                Document doc = null;
                try{
                domFactory = DocumentBuilderFactory.newInstance();
                domBuilder = domFactory.newDocumentBuilder();
  }
                catch(Exception exp){
                System.err.println(exp.toString());
                }

                // Create a new XML document and append the extracted portion to it
  doc = domBuilder.newDocument();
                Node tmpNode = doc.createElement("CompP2P");
  doc.appendChild(tmpNode);
                Element el = (Element)tmpNode.appendChild(doc.adoptNode(currentNode));

                return doc;

}

// Extract the portion of a XML tree corresponing to a specified Area
public Document extractGlobal(){
                Element[] nodes;
                NodeList groupnodes = newDoc.getElementsByTagName("Global");
                Element currentNode = null;

                int numgroups = groupnodes.getLength();
```

```
                        nodes = new Element[numgroups];
                        // 'For' control variable
                        boolean found = false;

                        // If only one element found
                        if (numgroups == 0){
                                // nothing (currently)
                        }else{
                                // find the desired group element
                                for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                                        nodes[i] = (Element)groupnodes.item(i);
                                        //if
(nodes[i].getAttributeNode("AreaID").getValue().equals(areaID)){
                                                currentNode = (Element) nodes[i];
                                        //}
                                }
                        }

                        Document doc = null;
                        try{
                        domFactory = DocumentBuilderFactory.newInstance();
                        domBuilder = domFactory.newDocumentBuilder();
                  }
                        catch(Exception exp){
                        System.err.println(exp.toString());
                        }

                        // Create a new XML document and append the extracted portion to it
                  doc = domBuilder.newDocument();
                        Node tmpNode = doc.createElement("CompP2P");
                  doc.appendChild(tmpNode);
                        Element el = (Element)tmpNode.appendChild(doc.adoptNode(currentNode));

                        return doc;

                }

                // Extract a element of the XML tree, specified in the String elementName
                // NOTE: Only works correctly if exists only one element with the passed name
                public Element extractElement(String elementName){
                        Node[] nodes;
                        NodeList groupnodes = newDoc.getElementsByTagName(elementName);
                        Node currentNode = null;

                        int numgroups = groupnodes.getLength();
                        nodes = new Element[numgroups];
                        // 'For' control variable
                        boolean found = false;

                        // If only one element found
                        if (numgroups == 1){
                                nodes[0] = (Element)groupnodes.item(0);
                                currentNode = nodes[0];
                        }else if (numgroups == 0){
                                // nothing (currently)
                        }else{
                                // nothing (currently)
                        }
                        Document doc = null;
```

```
                    try{
                    domFactory = DocumentBuilderFactory.newInstance();
                    domBuilder = domFactory.newDocumentBuilder();
          }

                    catch(FactoryConfigurationError exp){
                              System.err.println(exp.toString());
          }
          catch(ParserConfigurationException exp){
            System.err.println(exp.toString());
          }
                    catch(Exception exp){
            System.err.println(exp.toString());
          }

                    // Create a new XML document and append the extracted portion to it
          doc = domBuilder.newDocument();
          Node tmpNode = doc.createElement("CompP2P");
          doc.appendChild(tmpNode);
                    Element el = (Element)tmpNode.appendChild(doc.adoptNode(currentNode));

                    return el;

     }

     // Insert a new element in the XML tree
     public void insertNode(Element element){
                    Element tmpEl = (Element)
currentElement.appendChild(newDoc.adoptNode(element));
     }

     /*
     public void elementNode(String nodeID){
                    Element[] nodes;
                    NodeList groupnodes = newDoc.getElementsByTagName("Node");
                    int numgroups = groupnodes.getLength();
                    nodes = new Element[numgroups];
                    // 'For' control variable
                    boolean found = false;

                    // find the desired group element
                    for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                    nodes[i] = (Element)groupnodes.item(i);
                              if (nodes[i].getAttributeNode("id").getValue()==nodeID){
                                        currentElement = nodes[i];
                              }
                    }
     }
     */

     /*
     public void elementGroup(String groupID){
                    Element[] groups;
                    NodeList groupnodes = newDoc.getElementsByTagName("Group");
          int numgroups = groupnodes.getLength();
          groups = new Element[numgroups];
          // 'For' control variable
          boolean found = false;

                    // finthe desired group element
```

```
                for(int i = 0; ((i < numgroups)&&(found==false)); i++) {
                        groups[i] = (Element)groupnodes.item(i);
                                if (groups[i].getAttributeNode("id").getValue()==groupID){
                                        currentElement = groups[i];
                                }
                        }

                }
                */


                /*
                public String merge(String xmlString){
                        String str = new String();
                        try {

                                Transformer transformer =
TransformerFactory.newInstance().newTransformer(new StreamSource("xslt/merge-xml.xsl"));
                                //transformer.setParameter("second-xml",
"D:/Projects/merging_xml_files/second.xml");

                                //FileWriter mergedXMLWriter = new
FileWriter("D:/Projects/merging_xml_files/result.xml");
                                StringWriter sw=new StringWriter();
                                StringReader sr=new StringReader(xmlString);

                                Source src = new DOMSource(newDoc);
                                StreamSource ss = new StreamSource(sr);
                                StreamResult dest = new StreamResult(sw);

                                transformer.setParameter("second-xml", ss);

                                transformer.transform(src, dest);

                                //transformer.transform(new
StreamSource("D:/Projects/merging_xml_files/first.xml"), new StreamResult(mergedXMLWriter));
                                //mergedXMLWriter.close();
                        } catch (Exception e) {

                        }
                        return str;
                }
                */

                /*
                public Document extractNode(String elementName){
                        Node[] nodes;
                        NodeList groupnodes = newDoc.getElementsByTagName(elementName);
                        Node currentNode = null;

                        int numgroups = groupnodes.getLength();
                        nodes = new Element[numgroups];
                        // 'For' control variable
                        boolean found = false;

                        // If only one element found
                        if (numgroups == 1){
                                nodes[0] = (Element)groupnodes.item(0);
                                currentNode = nodes[0];
                                //System.out.println("goToElement: Only 1 found");
```

```
                        }else if (numgroups == 0){
                                // nothing (currently)
                                //System.out.println("goToElement: 0 found");
                        }else{
                                //System.out.println("goToElement: errrr!!");
                        }
                        Document doc = null;

                                try
                        {
                                domFactory = DocumentBuilderFactory.newInstance();
                                        domBuilder = domFactory.newDocumentBuilder();
                        }

                catch(FactoryConfigurationError exp)
                {
                   System.err.println(exp.toString());
                }
                catch(ParserConfigurationException exp)
                {
                   System.err.println(exp.toString());
                }
                catch(Exception exp)
                {
                   System.err.println(exp.toString());
                }

                doc = domBuilder.newDocument();
                //Element element1 = doc.createElement("CompP2P");
                //Node node1 = doc.createElement("CompP2P");

                Node tmpNode = doc.createElement("CompP2P");
                doc.appendChild(tmpNode);
                //Element el = (Element)tmpNode.appendChild(doc.createElement("CompP2P"));


                                Element el =
(Element)tmpNode.appendChild(doc.adoptNode(currentNode));
                        //Node node2 = doc.adoptNode(currentNode);
                        //el.appendChild(tmpNode);
                        //doc.appendChild(el);


                        return doc;
        }
        */

};
```

I.II. Herramienta CompP2PMonitor

```
/*
CompP2P platform

CompP2PMonitor monitorization application

Copyright (c) 2007 Hector Blanco de Frutos <hectorblanco@neopontec.com>
*/



/*******************************************************************************
 *    Base example code copyright
 * Copyright (c) 2000, 2004 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************
/

/*
 * Browser example snippet: bring up a browser
 *
 * For a list of all SWT example snippets see
 * http://www.eclipse.org/swt/snippets/
 *
 * @since 3.0
 */



import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.browser.*;

import java.net.Socket;

import org.w3c.dom.*;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;


import java.io.*;


//import org.w3c.dom.*;
//import org.xml.sax.*;
```

```
//import javax.xml.parsers.*;
import javax.xml.transform.*;
//import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;



public class CompP2PMonitor {
	public static void main(String [] args) throws Exception{



		// Peer connection data
		final String host="localhost";
		final int port=35557;

		final File appPath = new
File(System.getProperty("java.class.path"), ".");


		final String path = System.getProperty("user.dir") + "/";
		System.out.println("App Path: " + path);

		// Delete the XML data file if exists
		final File xmlFile = new File(path + "tmp/compP2Pdata.xml");
		xmlFile.delete();

		Display display = new Display();
		final Shell shell = new Shell(display);
		GridLayout gridLayout = new GridLayout();
		gridLayout.numColumns = 3;
		shell.setLayout(gridLayout);

		shell.setText("compP2P Monitor");

		ToolBar toolbar = new ToolBar(shell, SWT.NONE);
		ToolItem itemConnect = new ToolItem(toolbar, SWT.PUSH);
		itemConnect.setText("Get stats");
		ToolItem itemForward = new ToolItem(toolbar, SWT.PUSH);
		itemForward.setText("Shutdown");

		GridData data = new GridData();
		data.horizontalSpan = 3;
		toolbar.setLayoutData(data);

		data = new GridData();
		data.horizontalAlignment = GridData.FILL;
		data.horizontalSpan = 2;
		data.grabExcessHorizontalSpace = true;

		final Browser browser;
		try {
			browser = new Browser(shell, SWT.NONE);
		} catch (SWTError e) {
			System.out.println("Could not instantiate Browser:
" + e.getMessage());
			return;
		}
```

```
                    data = new GridData();
                    data.horizontalAlignment = GridData.FILL;
                    data.verticalAlignment = GridData.FILL;
                    data.horizontalSpan = 3;
                    data.grabExcessHorizontalSpace = true;
                    data.grabExcessVerticalSpace = true;
                    browser.setLayoutData(data);

                    final Label status = new Label(shell, SWT.NONE);
                    data = new GridData(GridData.FILL_HORIZONTAL);
                    data.horizontalSpan = 2;
                    status.setLayoutData(data);

                    final ProgressBar progressBar = new ProgressBar(shell,
SWT.NONE);
                    data = new GridData();
                    data.horizontalAlignment = GridData.END;
                    progressBar.setLayoutData(data);

                    /* event handling */
                    Listener listener = new Listener() {
                              public void handleEvent(Event event) {


                                        ToolItem item = (ToolItem)event.widget;
                                        String string = item.getText();
                                        int errorfound = 0;

                                        // Connecting button
                                        if (string.equals("Get stats")){
                                                  try{
                                                            // Remove the XML file
if it exists
                                                            xmlFile.delete();


          System.out.println("Connecting with the peer...");

          status.setText("Connecting with the peer...");
                                                            BufferedReader
reader=new BufferedReader(new InputStreamReader(System.in));

                                                            Socket socket = new
Socket (host, port);

          socket.setSoTimeout(15000);
                                                  DataOutputStream request = new
DataOutputStream (socket.getOutputStream());
                                                  request.writeUTF("stats");
                                                  request.flush();

                                                  DataInputStream reply = new
DataInputStream(socket.getInputStream());
                                                  //System.out.println(reply.readUTF
());
                                                  String replyText = new
String(reply.readUTF());

                                                  XMLCompP2P xmlP2P = new
```

```
XMLCompP2P(replyText);

xmlP2P.writeFile("tmp/compP2PdataArea.xml");

                                        if (!replyText.equals(null)){
                                         System.out.println("Stats
obtained");

          status.setText("Stats obtained");
                                        }

                                        request.close();
                                        reply.close();
                                        socket.close();


                                         }catch (Exception e) {
                                               System.out.println("Cannot
connect to the peer...");

System.out.println(e.toString());
                                               status.setText("Cannot
connect to the peer: " + e.toString());
                                               browser.setUrl(path +
"data/errorpage.html");
                                               errorfound = 1;
                                    }

                                         try{
                                                  if (errorfound == 0){
                                                   TransformerFactory
transformerFactory = TransformerFactory.newInstance();

                                                  StreamSource source =
                                                   new StreamSource( path
+ "tmp/compP2PdataArea.xml");
                                                  StreamResult result =
                                                   new StreamResult(path
+ "tmp/compP2PStatsArea.html");
                                                  StreamSource style =
                                                   new StreamSource(path
+ "data/compP2P-stats-area-1.0.xsl");

                                                  Transformer transformer
= transformerFactory.newTransformer(style);


transformer.transform(source, result);
                                                  browser.setUrl(path +
"tmp/compP2PStatsArea.html");
                                                   }

                                  } catch (Exception e) {
                                        e.printStackTrace();
                                        browser.setUrl(path +
"data/errorpage.html");
                                        errorfound = 1;
                                  }
```

```
                                        //------------------------------
-------------------------
                                        //------------------------------
-------------------------
                                }
                                else if (string.equals("Shutdown")){
                                        try{
                                                // Remove the XML file
if it exists
                                                xmlFile.delete();


        System.out.println("Connecting with the peer...");

        status.setText("Connecting with the peer...");
                                                BufferedReader
reader=new BufferedReader(new InputStreamReader(System.in));

                                                Socket socket = new
Socket (host, port);
                                        DataOutputStream request = new
DataOutputStream (socket.getOutputStream());
                                        request.writeUTF("shutdown");
                                        request.flush();

                                        System.out.println("Shutdown
command sent");

        status.setText("Shutdown command sent");

                                        request.close();
                                        socket.close();

                                                //browser.setUrl(System
.getProperty("user.dir") + "/compP2Pdata.xml");
                                         }catch (Exception e) {
                                               System.out.println("Cannot
connect to the peer...");
                                               status.setText("Cannot
connect to the peer...");
                                               browser.setUrl(path +
"data/errorpage.html");
                                    }
                                        //------------------------------
-------------------------
                                        //------------------------------
-------------------------

                                }
                                else if (string.equals("Stop")){
                                        browser.stop();
                                }
                                else if (string.equals("Refresh")){
                                        browser.refresh();
                                }
                                else if (string.equals("Go")){
                                        //browser.setUrl(location.getText
());
```

```
                                        }
                                }
                        };
                        browser.addProgressListener(new ProgressListener() {
                                public void changed(ProgressEvent event) {
                                                if (event.total == 0) return;
                                                int ratio = event.current * 100 /
event.total;
                                                progressBar.setSelection(ratio);
                                }
                                public void completed(ProgressEvent event) {
                                        progressBar.setSelection(0);
                                }
                        });
                        browser.addStatusTextListener(new StatusTextListener() {
                                public void changed(StatusTextEvent event) {
                                        //status.setText(event.text);
                                }
                        });
                        browser.addLocationListener(new LocationListener() {
                                public void changed(LocationEvent event) {

                                }
                                public void changing(LocationEvent event) {
                                }
                        });
                        itemConnect.addListener(SWT.Selection, listener);
                        itemForward.addListener(SWT.Selection, listener);

                        shell.open();


                        while (!shell.isDisposed()) {
                                if (!display.readAndDispatch())
                                        display.sleep();
                        }
                        display.dispose();
                }
}
```

I.III. CompP2PResident

```
import java.net.Socket;
import org.w3c.dom.*;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;




public class residentMonitor {

public static final String DATE_FORMAT_NOW = "MM/dd/yyyy - HH:mm:ss.SSSS";
public static String host="localhost";
public static int port=35557;

public static void main(String [] args) throws Exception{
  boolean done = false;

  Vector statsTime = new Vector();
  Vector freeMemory = new Vector();
  Vector NumberOfPeers = new Vector();


  int counter = 0;
  long groupFreeMemory = 0;
  int groupNumberOfPeers = 0;




  while(done == false){
         System.out.println("[" + getTime(DATE_FORMAT_NOW) + "]Requesting
stats...");

         try{
                  Socket socket = new Socket (host, port);
                  socket.setSoTimeout(30000);
                  DataOutputStream request = new DataOutputStream
(socket.getOutputStream());

                  request.writeUTF("stats");
                  request.flush();

                  DataInputStream reply = new
```

```
DataInputStream(socket.getInputStream());
                    String replyText = new String(reply.readUTF());
                    XMLCompP2P xmlP2P = new XMLCompP2P(replyText);
                    xmlP2P.writeFile("/var/www/compP2PWeb/xml/" +
getTime("yyyy.MM.dd-HH.mm.ss") + ".xml");



                    System.out.println("/var/www/compP2PWeb/xml/" +
getTime("yyyy.MM.dd-HH.mm.ss") + ".xml");

                    System.out.println("[" + getTime(DATE_FORMAT_NOW) + "] Stats
obtained.");

                    request.close();
                    reply.close();
                    socket.close();

          }catch(Exception e){
                    System.out.println("[ERROR] Can't connect");
                    System.out.println(e.toString());
          }
    try {
           Thread.sleep(30000);
    } catch(Exception e) {
   }
 }
}

  public static String getTime(String format){
          Calendar cal = Calendar.getInstance();
          SimpleDateFormat sdf = new SimpleDateFormat(format);
          return sdf.format(cal.getTime());
  }
}
```

I.IV. CompP2PStatsRequester

```
import java.net.Socket;

import org.w3c.dom.*;

import java.security.*;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.*;

import java.io.*;
import java.io.File;
import java.util.Arrays;
import java.text.*;

import javax.xml.transform.*;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;


/*
Arguments:
0 = Action
          - peerlist
          - chart
          - stats
1 = Number of XML files to read from
2 = Domain (String)
          - global
          - area
                    - 3: area ID
          - peer
                    - 3: peer ID
4 = Field (or fields in future versions)

...

5 = -p (path command indicator)

6 = Path

*/


public class CompP2PstatsRequester {
```

```
public static final String DATE_FORMAT_NOW = "HH:mm-MM-dd-yyyy";

          static public String xmlPath = "/var/www/compP2PWeb/xml";
          static public String targetPath = "/var/www/compP2PWeb";

 public static void main(String [] args) throws Exception{
          int numFiles = 0;
          String action, field, domain, objectID;
          field = new String();
          domain = new String();
          objectID = new String();
          action = new String();
          XMLCompP2P xmlObjects[];
          String peerList[];


          // Parsing arguments
          if (args.length > 0) {
                    try {
                              action = args[0];
                    } catch (NumberFormatException e) {
                              System.err.println("First argument must be a string
(Action)");
                              System.exit(1);
                    }
                    if (action.equals("chart")){

                              try {
                                        numFiles = Integer.parseInt(args[1]);
                              } catch (NumberFormatException e) {
                                        System.err.println("Second argument must
be an integer (number of XML files)");
                                        System.exit(1);
                              }
                              /*
                              try {
                                        field = args[2];
                              } catch (NumberFormatException e) {
                                        System.out.println("Third argument must be
a String (field to work with)");
                                        System.exit(1);
                              }
                              */

                              try {
                                        domain = args[2];
                              } catch (NumberFormatException e) {
                                        System.out.println("Third argument must be
a String (domain)");
                                        System.exit(1);
                              }

                              try {
                                        String str = args[3];
                                        if (str.equals("-p")){
                                                  targetPath = args[4];
                                        }
                              } catch (NumberFormatException e) {
                                        System.out.println("Path argument not
```

```
passed correctly");
                                        System.exit(1);
                                }
                                // Args parsed


                                if ( domain.equals("global")){
                                        xmlObjects = readFiles(numFiles, domain,
args[3], "");
                                        extractGlobal(xmlObjects, args[3]);

                                }else if( domain.equals("area")){
                                        xmlObjects = readFiles(numFiles, domain,
args[4], args[3]);
                                        extractArea(xmlObjects, args[3], args[4]);

                                }else if( domain.equals("peer")){
                                        xmlObjects = readFiles(numFiles, domain,
args[4], args[3]);
                                        extractPeer(xmlObjects, args[3], args[4]);
                                }
                        }else if (action.equals("peerlist")){
                                try {
                                        String str = args[1];
                                        if (str.equals("-p")){
                                                targetPath = args[2];
                                        }
                                } catch (NumberFormatException e) {
                                        System.out.println("Path argument not
passed correctly");
                                        System.exit(1);
                                }
                                writePeersList();
                        }else if (action.equals("arealist")){
                                try {
                                        String str = args[1];
                                        if (str.equals("-p")){
                                                targetPath = args[2];
                                        }
                                } catch (NumberFormatException e) {
                                        System.out.println("Path argument not
passed correctly");
                                        System.exit(1);
                                }
                                writeAreaList();
                        }else if (action.equals("stats")){
                                try {
                                        numFiles = Integer.parseInt(args[1]);
                                } catch (NumberFormatException e) {
                                        System.err.println("Second argument must
be an integer (number of XML files)");
                                        System.exit(1);
                                }

                                try {
                                        domain = args[2];
                                } catch (NumberFormatException e) {
                                        System.out.println("Third argument must be
a String (domain)");
```

```
                                        System.exit(1);
                                }
                                if (domain.equals("peer")){
                                        generatePeerStats(args[3]);
                                }else if (domain.equals("area")){
                                        generateAreaStats(args[3]);
                                }
                        }

                }else{
                        System.out.println("No arguments passed.");
                        System.exit(1);
                }


        }

        static public XMLCompP2P[] readFiles(int numFiles, String domain, String field,
String objID){
                File path = new File(targetPath + "/xml");
                XMLCompP2P xmlObjects[];
                String times[];
                int j =0, filesToProcess = 0;
                String md5String = new String();

                if (domain.equals("global")){
                        md5String = "_";
                }else{
                        md5String = "_" + md5(objID) + "_";
                }

                // List contents of the dir
                File files[];
                files = path.listFiles();
                if (files.length > 200){ filesToProcess = 200; }else{ filesToProcess =
files.length; }
                Arrays.sort(files);

                String finalFiles[];

                String lastFile = new String();
                try{
                        BufferedReader in = new BufferedReader(new FileReader("tmp/"
+ domain + md5String + field + "_lastFile.dat"));
                        String str;
                        //if ((str = in.readLine()) != null) {
                                str = in.readLine();
                                lastFile = str;
                        //}
                        in.close();
                        System.out.println("Last file processed: " + str);
                }
                catch (Exception ee){
                        System.out.println(ee.toString());
                        lastFile = files[0].toString();
                        System.out.println("Set last file to: " + lastFile);
                }

                boolean found = false;
```

```
          int i;
          System.out.println("Comparando...");
          i = 0;
          while((i<filesToProcess)&&(found==false)){
                    //for (i=0; ((i< files.length)&&(found=true)); i++){
                              //System.out.println("\t" + files[i].toString() + "
- " + lastFile + "<br>");
                              if (lastFile.equals(files[i].toString())){
                                        found = true;
                                        System.out.println("Eureka!<br>");
                                        i--;
                              }
                              i++;
          }
          System.out.println("Indice del archivo: " + i + " - Archivos: " +
files.length + "<br>");

          finalFiles = new String[filesToProcess-i];
          j = i;
          for (i=0; j< filesToProcess; j++){
                    finalFiles[i] = files[j].toString();
                    i++;
          }
          System.out.println("Archivos finales: " + finalFiles.length);

          //if (files.length>numFiles){
          j = 0;
          xmlObjects = new XMLCompP2P[finalFiles.length];
          times = new String[finalFiles.length];
          boolean lastFound = false;
          for (i=0; i< finalFiles.length; i++){
                              xmlObjects[j] = new XMLCompP2P();

          xmlObjects[j].readFromFile(finalFiles[i].toString());
                              times[j] =
getTimeFromFileName(finalFiles[i].toString());
                              xmlObjects[j].goToElement("CompP2P");
                              xmlObjects[j].addP2PAttribute("TimeAttr",
times[j]);
                              j++;
          }

          try{
                    // Write the name of the last file processed.
                    FileOutputStream fout2 = new FileOutputStream ("tmp/"+domain
+ md5String + field + "_" + "lastFile.dat");


                    new PrintStream(fout2).println(finalFiles[j-1]);

          }
          catch (Exception e){
                    System.out.println(e.toString());
          }

          return xmlObjects;
 }

static void extractPeer(XMLCompP2P xmlObjects[], String peerID, String field){
```

```
          System.out.println("Obtain data from peer");
          Document doc;
          XMLCompP2P xmlPeer = new XMLCompP2P();
          String time;
          // File action
          try
          {
                    FileOutputStream fout = new FileOutputStream ("tmp/peer_" +
md5(peerID) + "_" +field + ".dat", true);
                    for (int i=0; i< xmlObjects.length; i++){
                              try
                              {
                                        xmlObjects[i].goToElement("CompP2P");
                                        time =
xmlObjects[i].getAttribute("TimeAttr");

                                        doc = xmlObjects[i].extractPeer(peerID);
                                        xmlPeer = new XMLCompP2P(doc);
                                        xmlPeer.goToElement("Peer");

                                        new PrintStream(fout).println ( time +
"\t" + xmlPeer.getAttribute(field) + "\t" );
                              }catch (Exception e){
                                        System.out.println(e.toString());
                              }
                    }
          }catch (Exception e){
                    System.out.println("Error writing data file");
                    System.out.println(e.toString());
          }
 }

 static void  extractArea(XMLCompP2P xmlObjects[], String areaID, String field){
          System.out.println("Obtain data from Area");
          Document doc;
          XMLCompP2P xmlPeer = new XMLCompP2P();
          String time;
          // File action

          try{
                    FileOutputStream fout = new FileOutputStream ("tmp/area_" +
md5(areaID) + "_" + field + ".dat", true);
                    for (int i=0; i< xmlObjects.length; i++){
                              try
                              {

          System.out.println(xmlObjects[i].toString());
                                        xmlObjects[i].goToElement("CompP2P");
                                        time =
xmlObjects[i].getAttribute("TimeAttr");

                                        doc = xmlObjects[i].extractArea(areaID);
                                        xmlPeer = new XMLCompP2P(doc);
                                        xmlPeer.goToElement("ManagerInfo");

                                        new PrintStream(fout).println ( time +
"\t" + xmlPeer.getAttribute(field) + "\t" );
                              }catch (Exception e){
```

```
                                        System.out.println(e.toString());
                                }
                        }
                }catch (Exception e){
                        System.out.println("Error writing data file");
                        System.out.println(e.toString());
                }



 }

 static void  extractGlobal(XMLCompP2P xmlObjects[], String field){

                //System.out.println("Obtain data from Global");
                Document doc;
                XMLCompP2P xmlPeer = new XMLCompP2P();
                String time = new String();
                // File action
                try
                {
                        FileOutputStream fout = new FileOutputStream ("tmp/global_" +
field + ".dat", true);
                        //new PrintStream(fout).println("Time\t*Data");

                        int i;
                        for (i=0; i< xmlObjects.length; i++){

                                xmlObjects[i].goToElement("CompP2P");
                                time = xmlObjects[i].getAttribute("TimeAttr");

                                doc = xmlObjects[i].extractGlobal();
                                xmlPeer = new XMLCompP2P(doc);
                                xmlPeer.goToElement("Global");

                                String val = xmlPeer.getP2PAttribute(field);
                                //if (val.equals("")){ val = "0"; time
="00:00-00-00-1970";}

                                new PrintStream(fout).println ( time + "\t" + val +
"\t" );
                        }


                        fout.close();

                }catch (Exception e){
                        System.out.println("Error writing data file");
                        System.out.println(e.toString());
                }
 }

 static void generatePeerStats(String peerID){
                File path = new File(targetPath + "/xml");
                System.out.println("Obtain stats from peer");
                Document doc;
                XMLCompP2P xmlPeer = new XMLCompP2P();
                String time;
                int j;
```

```
        // List contents of the dir
        File files[];
        files = path.listFiles();
        Arrays.sort(files);

        // File action
        try
        {
                if (files.length>0){
                        j = 0;
                        XMLCompP2P xmlObject = new XMLCompP2P();


        xmlObject.readFromFile(files[files.length-1].toString());
                        xmlObject.goToElement("CompP2P");
                        System.out.println(peerID);
                        doc = xmlObject.extractPeer(peerID);
                        //peerList = xmlObject.getPeerList();
                        xmlPeer = new XMLCompP2P(doc);

                        xmlPeer.writeFile("xmlPeer.xml");

                        // Do a XSLT transformation
                        TransformerFactory transformerFactory =
TransformerFactory.newInstance();

                        StreamSource source = new
StreamSource("xmlPeer.xml");
                        StreamResult result = new
StreamResult("tmp/peerStats.html");
                        StreamSource style = new
StreamSource( "data/compP2P-stats-peer-1.0.xsl");

                        Transformer transformer =
transformerFactory.newTransformer(style);

                        transformer.transform(source, result);
                }

        }catch (Exception e){
                System.out.println("Error generating stats from a peer");
                System.out.println(e.toString());
        }



 }

 static void generateAreaStats(String areaID){
        File path = new File(targetPath + "/xml");
        //System.out.println("Obtain stats from area");
        Document doc;
        XMLCompP2P xmlArea = new XMLCompP2P();
        String time;
        int j;

        // List contents of the dir
        File files[];
```

```
          files = path.listFiles();
          Arrays.sort(files);

          // File action
          try
          {
                    if (files.length>0){
                              j = 0;
                              XMLCompP2P xmlObject = new XMLCompP2P();
                              //times = new String[numFiles];
                              //System.out.println("Last XML file: " +
files[files.length-1].toString());

          xmlObject.readFromFile(files[files.length-1].toString());
                              xmlObject.goToElement("CompP2P");

                              doc = xmlObject.extractArea(areaID);

                              //peerList = xmlObject.getPeerList();
                              xmlArea = new XMLCompP2P(doc);

                              xmlArea.writeFile("xmlArea.xml");

                              // Do a XSLT transformation
                              TransformerFactory transformerFactory =
TransformerFactory.newInstance();

                              StreamSource source = new
StreamSource("xmlArea.xml");
                              StreamResult result = new
StreamResult("tmp/areaStats.html");
                              StreamSource style = new
StreamSource( "data/compP2P-stats-area-1.0.xsl");

                              Transformer transformer =
transformerFactory.newTransformer(style);

                              transformer.transform(source, result);
                    }

          }catch (Exception e){
                    System.out.println("Error generating stats from a peer");
                    System.out.println(e.toString());
          }



 }

static String getTimeFromFileName(String filename){

           String Str = new String();
           char dest [  ]  = new char [ filename.length()-3 ] ;
           filename.getChars ( filename.length()-23, filename.length()-4 , dest,
0 ) ;

          Date a = new Date();

          DateFormat dfm = new SimpleDateFormat("yyyy.MM.dd-HH.mm.ss");
```

```
          try{
                    Str = new String(dest);
                    a = dfm.parse(Str);
          }catch(Exception e){
                    System.out.println("\nError generating date");
                    System.out.println(e.toString());
          }

          Calendar cal = Calendar.getInstance();
          SimpleDateFormat sdf = new SimpleDateFormat("HH:mm-MM-dd-yyyy");

          return sdf.format(a);
}


static void writePeersList(){
          File path = new File(targetPath + "/xml");
          int j =0;

          XMLCompP2P xmlObject;
          String peerList[] = new String[1];

          // List contents of the dir
          File files[];
          files = path.listFiles();
          Arrays.sort(files);
          if (files.length>0){
                    j = 0;
                    xmlObject = new XMLCompP2P();
                    //times = new String[numFiles];
                    //System.out.println("Last XML file: " +
files[files.length-1].toString());
                    xmlObject.readFromFile(files[files.length-1].toString());
                    xmlObject.goToElement("CompP2P");
                    peerList = xmlObject.getPeerList();
          }

          try{
                    //System.out.println(targetPath + "/peerlist.dat");
                    FileOutputStream fout = new FileOutputStream (targetPath +
"/tmp/peerlist.dat");
                    for(j = 0; j < peerList.length; j++){
                              //System.out.println(peerList[j]);
                              new PrintStream(fout).println ( peerList[j] );
                    }
          }catch (Exception e){
                    System.out.println("Error writing data file");
                    System.out.println(e.toString());
          }


}

static void writeAreaList(){
          File path = new File(targetPath + "/xml");
          int j =0;

          XMLCompP2P xmlObject;
          String peerList[] = new String[1];
```

```
          // List contents of the dir
          File files[];
          files = path.listFiles();
          Arrays.sort(files);
          if (files.length>0){
                    j = 0;
                    xmlObject = new XMLCompP2P();
                    //times = new String[numFiles];
                    //System.out.println("Last XML file: " +
files[files.length-1].toString());
                    xmlObject.readFromFile(files[files.length-1].toString());
                    xmlObject.goToElement("CompP2P");
                    peerList = xmlObject.getAreaList();
          }

          try{
                    //System.out.println(targetPath + "/arealist.dat");
                    FileOutputStream fout = new FileOutputStream (targetPath +
"/tmp/arealist.dat");
                    for(j = 0; j < peerList.length; j++){
                              //System.out.println(peerList[j]);
                              new PrintStream(fout).println ( peerList[j] );
                    }
          }catch (Exception e){
                    System.out.println("Error writing data file");
                    System.out.println(e.toString());
          }


}


// Calcula el MD5 de una cadena de texto
static String md5 (String sessionid) {
          /*
          try{
                    byte[] defaultBytes = sessionid.getBytes();

                    MessageDigest algorithm = MessageDigest.getInstance("MD5");
                    algorithm.reset();
                    algorithm.update(defaultBytes);
                    byte messageDigest[] = algorithm.digest();

                    StringBuffer hexString = new StringBuffer();
                    for (int i=0;i<messageDigest.length;i++) {
                              hexString.append(Integer.toHexString(0xff &
messageDigest[i]));
                    }
                    String foo = messageDigest.toString();
                    System.out.println("sessionid "+sessionid+" md5 version is
"+hexString.toString());
                    sessionid=hexString+"";
                    return sessionid;

          }catch(NoSuchAlgorithmException nsae){
                    return "";
          }
          */
```

```
            return sessionid.substring(14);
}

}
```

II. Índice de ilustraciones

III. Referencias

[GR06]: Íñigo Goiri Presa, Josep María Rius Torrentó: *CompP2P: Sistema de Computació distribuïda Peer to Peer. 2006*

[JXP]: JXTA v2.0 Protocols Specification. Sun Microsystems. https://jxta-spec.dev.java.net/nonav/JXTAProtocols.html

[PEER02]: Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins 2 ,Zhichen Xu: *Peer-to-Peer Computing*. HPL-2002-57. HP Laboratories Palo Alto. 2002.

[WSD01]: Web Services Description Language (WSDL) 1.1. Copyright© 2001 Ariba, International Business Machines Corporation, Microsoft

IV. Bibliografía

1. CompP2P: Sistema de Computació Distribuïda Peer to Peer
   Íñigo Goiri Presa
   Josep Maria Rius Torrentó

2. Artículo de JXTA en Wikipedia
   http://en.wikipedia.org/wiki/JXTA

3. Página web oficial del proyecto JXTA
   http://www.jxta.org

4. Página de JXTA en Sun Microsystems
   http://www.sun.com/software/jxta/